Red Hat JBoss Fuse Service Works 6.0

Getting Started Guide

Use this guide to learn how to produce results quickly and efficiently.

Last Updated: 2017-11-12

Red Hat JBoss Fuse Service Works 6.0 Getting Started Guide

Use this guide to learn how to produce results quickly and efficiently.

Red Hat Content Services

Legal Notice

Abstract

This guide provides developers with an introduction to JBoss Fuse Service Works.

Table of Contents

CHAPTER 1. RED HAT JBOSS FUSE SERVICE WORKS

1.1. WHAT IS RED HAT JBOSS FUSE SERVICE WORKS?

Red Hat JBoss Fuse Service Works is a platform for developing enterprise application integration (EAI) and service-oriented architecture (SOA) solutions. It consists of a service component framework, business rules/complex event processing, life-cycle governance, runtime governance and process automation. It is built on the same core as JBoss Fuse with enterprise messaging, Camel and CXF so, therefore, users can use it to design, deploy, integrate and orchestrate business services.

Report a bug

1.2. CORE CAPABILITIES

The benefits of using Red Hat JBoss Fuse Service Works include the following:

Enterprise Integration Pattern (EIP) Based Development

The versatile EIP framework is implemented in routing and transformation processes for faster and more efficient integration solutions.

High Performance Messaging

A high performance messaging broker supports messaging patterns such as publish-subscribe, point-to-point and store-forward, and multiple cross language clients.

Service Development

The web services framework exposes integration assets as services and calls external services, supporting all major web services standards. It also supports RESTful calls.

Structured Service Development

A lightweight service development framework provides full lifecycle support for developing, deploying, and managing service-based applications.

Automatable Registry with Workflow

Manage the lifecycle of services from design, development and deployment by defining, exposing and enforcing rules or policies.

Business Transaction Monitoring

Capture service activity information, define and collect metrics, and define alerts and SLAs.

Report a bug

1.3. SYSTEM INTEGRATION

Integrating your major business systems into a cohesive infrastructure can be a challenge, especially when you have legacy applications. Red Hat JBoss Fuse Service Works has a number of ways enable you to integrate both new and legacy applications. Development is simplified with a transparent, lightweight service framework which uses EIP technology. This allows developers to focus on higher order concepts while still working with familiar technologies such as Apache Camel, BPEL, BPMN or POJOs. To reduce the operational costs of production and maintenance, the platform utilizes

automatable, content-aware repository and service activity monitoring. These support the entire service lifecycle and development, QA and production teams with run-time and design-time visibility, monitoring and alerting.

Report a bug

1.4. INTEGRATION USE CASE

Acme Equity is a large financial service. The company possesses many databases and systems. Some are older, COBOL-based legacy systems and some are databases obtained through the acquisition of smaller companies in recent years. It is challenging and expensive to integrate these databases as business rules frequently change. The company wants to develop a new series of client-facing e-commerce websites, but these may not synchronize well with the existing systems as they currently stand.

The company wants an inexpensive solution but one that will adhere to the strict regulations and security requirements of the financial sector. What the company does not want to do is to have to write and maintain “glue code” to connect their legacy databases and systems.

Red Hat JBoss Fuse Service Works was selected as a middleware layer to integrate these legacy systems with the new customer websites. It provides a bridge between front-end and back-end systems. Business rules implemented with Red Hat JBoss Fuse Service Works can be updated quickly and easily.

As a result, older systems can now synchronize with newer ones due to the unifying methods of Red Hat JBoss Fuse Service Works. There are no bottlenecks, even with tens of thousands of transactions per month. Various integration types, such as XML, JMS and FTP, are used to move data between systems. Any one of a number of enterprise-standard messaging systems can be plugged into Red Hat JBoss Fuse Service Works providing further flexibility.

An additional benefit is that the system can now be scaled upwards easily as more servers and databases are added to the existing infrastructure.

Report a bug

1.5. UTILIZING RED HAT JBOSS FUSE SERVICE WORKS IN A BUSINESS ENVIRONMENT

Cost reduction can be achieved due to the implementation of services that can quickly communicate with each other with less chance of error messages occurring. Through enhanced productivity and sourcing options, ongoing costs can be reduced.

Information and business processes can be shared faster because of the increased connectivity. This is enhanced by web services, which can be used to connect clients easily.

Legacy systems can be used in conjunction with the web services to allow different systems to "speak the same language". This reduces the amount of upgrades and custom code required to make systems synchronize.

Report a bug

CHAPTER 2. READ ME

2.1. BACK UP YOUR DATA

WARNING

Red Hat recommends that you back up your system settings and data before undertaking any of the configuration tasks mentioned in this book.

Report a bug

2.2. RED HAT DOCUMENTATION SITE

Red Hat's official documentation site is at https://access.redhat.com/site/documentation/. There you will find the latest version of every book, including this one.

Report a bug

2.3. EAP_HOME

`EAP_HOME` refers to the root directory of the Red Hat JBoss Enterprise Application Platform installation on which JBoss Fuse Service Works is deployed.

Report a bug

2.4. MODE

`MODE` will either be **`standalone`** or **`domain`** depending on whether your instance of JBoss Enterprise Application Platform is running in standalone or domain mode. Substitute one of these whenever you see **`MODE`** in a file path in this documentation.

Report a bug

CHAPTER 3. DOWNLOADING RED HAT JBOSS FUSE SERVICE WORKS

3.1. ABOUT THE RED HAT CUSTOMER PORTAL

The *Red Hat Customer Portal* is the centralized platform for Red Hat knowledge and subscription resources. Use the *Red Hat Customer Portal* to:

- Manage and maintain Red Hat entitlements and support contracts;
- Download officially-supported software;
- Access product documentation and the Red Hat Knowledgebase;
- Contact Global Support Services; and
- File bugs against Red Hat products.

The Customer Portal is available here: https://access.redhat.com.

Report a bug

3.2. JBOSS FUSE SERVICE WORKS 6.0.0 PACKAGES FOR DOWNLOAD

Table 3.1.

Package	Description
Red Hat JBoss Fuse Service Works 6.0.0 Javadocs for Run-Time Governance	Javadocs for Run-Time Governance.
Red Hat JBoss Fuse Service Works 6.0.0 Javadocs for S-RAMP	Javadocs for S-RAMP.
Red Hat JBoss Fuse Service Works 6.0.0 Javadocs for SwitchYard	Javadocs for SwitchYard.
Red Hat JBoss Fuse Service Works 6.0.0 Maven Repository	Red Hat provides a Maven repository containing artifacts required to build applications for Red Hat JBoss Fuse Service Works. This package enables users to setup the repository offline.
Red Hat JBoss Fuse Service Works 6.0.0 Source Code	The source code package contains the complete source code for the Red Hat JBoss Fuse Service Works product.
Red Hat JBoss Fuse Service Works 6.0.0 Installer	Red Hat provides this tool to simplify product installation. The installer can be run with a graphical interface or from a script on the command line.

Report a bug

3.3. DOWNLOAD FILES FROM THE RED HAT CUSTOMER PORTAL

Prerequisites

- Before you begin this task, you need a Customer Portal account. Browse to https://access.redhat.com and click the **Register** link in the upper right corner to create an account.

Procedure 3.1. Log in and Download Files from the Red Hat Customer Portal

1. Browse to https://access.redhat.com and click the **Log in** link in the top right corner. Enter your credentials and click **Log In**.

   Result

   You are logged into RHN and you are returned to the main web page at https://access.redhat.com.

2. **Navigate to the Downloads page.**
   Use one of the following options to navigate to the **Downloads** page.

   - Click the **Downloads** link in the top navigation bar.
   - Navigate directly to https://access.redhat.com/downloads/.

3. **Select the product and version to download.**
   Use one of the following ways to choose the correct product and version to download.

   - Step through the navigation one level at a time.
   - Search for your product using the search area at the top right-hand side of the screen.

4. **Download the appropriate file for your operating system and installation method of choice.**
   Depending on the product you choose, you may have the choice of a Zip archive, RPM, or native installer for a specific operating system and architecture. Click either the file name or the **Download** link to the right of the file you want to download.

Result

The file is downloaded to your computer.

Report a bug

3.4. CHECKSUM VALIDATION

Checksum validation is used to ensure a downloaded file has not been corrupted. Checksum validation employs algorithms that compute a fixed-size datum (or checksum) from an arbitrary block of digital data. If two parties compute a checksum of a particular file using the same algorithm, the results will be identical. Therefore, when computing the checksum of a downloaded file using the same algorithm as the supplier, if the checksums match, the integrity of the file is confirmed. If there is a discrepancy, the file has been corrupted in the download process.

Report a bug

3.5. VERIFY DOWNLOADED FILES

Procedure 3.2. Verify File Checksums on Red Hat Enterprise Linux

1. **Obtain checksum values for the downloaded file**
   a. Navigate to https://access.redhat.com/jbossnetwork/. Login if required.
   b. Select your **`Product`** and **`Version`**.
   c. Select the package you want to verify to navigate to the **`Software Details`** page.
   d. Take note of the **`MD5`** and **`SHA-256`** checksum values.
2. **Run a checksum tool on the file**
   a. Navigate to the directory containing the downloaded file in a terminal window.
   b. Run **`md5sum`** ***`downloaded_file`***.
   c. Run **`sha256sum`** ***`downloaded_file`***.

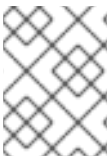
   Example output:

```
[localhost]$ md5sum downloaded_file
4564d1a5190110dbe8170e50d7353a97  downloaded_file
[localhost]$ sha256sum downloaded_file
25b6bd3c5f47b316639b014d041cdb6a515e3a4a32d30a479141cd8ceecb853e
downloaded_file
```

3. **Compare the checksum values**
   a. Compare the checksum values returned by the **`md5sum`** and **`sha256sum`** commands with the corresponding values displayed on the **`Software Details`** page.
   b. Download the file again if the two checksum values are not identical. A difference between the checksum values indicates that the file has either been corrupted during download or has been modified since it was uploaded to the server. Contact Red Hat Support for assistance if after several downloads the checksum does not successfully validate.
   c. The downloaded file is safe to use if the two checksum values are identical.

Result

The integrity of the downloaded file is verified.



NOTE

No checksum tool is included with **Microsoft Windows**. Download a third-party MD5 application such as **MD5summer** from http://www.md5summer.org/.

Report a bug

CHAPTER 4. INSTALLING RED HAT JBOSS FUSE SERVICE WORKS

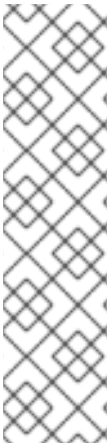
4.1. INSTALLING JBOSS FUSE SERVICE WORKS BY GUI

Prerequisites

- You must have already downloaded the Red Hat JBoss Fuse Service Works installer file from Red Hat Customer Portal.
- You must have a supported JDK installed. See Red Hat JBoss Fuse Service Works Supported Configurations.
- If you would like to configure use of the online Maven repository during this installation, you must have Maven already installed.

IMPORTANT

The set of online remote repositories is a technology preview source of components. As such, it is not in scope of patching and is supported only for use in development environment. Using the set of online repositories in production environment is a potential source of security vulnerabilities and is therefore not a supported use case. For more information see https://access.redhat.com/site/maven-repository.



NOTE

If you have multiple versions of Java installed on your server, Red Hat recommends setting PATH, JAVA_HOME, and JBOSS_HOME to ensure the correct version of Java is used at runtime:

- Set **`JAVA_HOME`** to the version of Java. For example, **`/usr/lib/jvm/java-1.7.0-openjdk-1.7.0.`*`VERSION`*`/`**.
- Set **`PATH`** to include the same version of Java.
- Set **`JBOSS_HOME`** to your **`jboss-eap-6.1`** directory.

Procedure 4.1. Install JBoss Fuse Service Works

1. Open a new terminal and navigate to the folder where you have downloaded the JBoss Fuse Service Works installer file.
2. Run the following command, replacing *6.0.0.GA-redhat-4* for the version you downloaded if required:

```
java -jar jboss-fsw-installer-6.0.0.GA-redhat-4.jar
```

3. You are presented with the End User License Agreement. If you agree with the terms, select **`I accept the terms of this license agreement`** and then select **`Next`** to continue.

4. You are prompted for the location in which you want to install JBoss Fuse Service Works. Specify a location and select **Next**. You will be notified that an instance of JBoss EAP will be placed in the chosen location. Select **OK** to proceed.

WARNING

A warning message is displayed if the directory already exists. Red Hat does not recommend installing over the top of an existing installation. However, if you wish to proceed, select **Yes** to continue.

5. You are prompted for the installation packs to install. Select **Next** to continue with the default options.

NOTE

Depending on your deployment, you can select packs other than the default options. For example, if you do not wish to install every pack, you could select SwitchYard and Run-Time Governance Client only, for a basic installation.

6. You are prompted to create an administrative user. Once created, this user will be added to the ManagementRealm and can be used to access the Management Console and other applications secured using ManagementRealm. Enter the new username and password in the appropriate fields and select **Next**.

7. You are prompted to create a governance user. It will give you access to S-RAMP, DTGov and RTGov consoles along with the S-RAMP Command Line Interface. Enter a username and password and select **Next**.

8. You are then prompted to setup your Maven repository. This will enable you to build the quickstarts provided. Specify the location of an existing or new location for your Maven settings file (typically `~/.m2/settings.xml`). If you do not currently have a settings file, a new `settings.xml` file will be created for you in the location you provide. Select **Next**.

NOTE

If you choose to skip the maven repository setup, you will be asked to confirm this decision. Select **Yes** to continue.

9. You are prompted to configure the SAML keystore. It is used by a Java KeyStore to sign authentication tokens. Enter the SAML keystore password and select **Next**.

10. You are then prompted to enable the Java Security Manager. It enforces access rules at the JVM runtime based on one or more security polices. Select **Next** when you are ready to proceed.

IMPORTANT

Enabling the Java Security Manager may reduce server performance.

11. At this point you are asked to choose between the default or advanced configuration. Select the default configuration for a basic configuration, and select **Next** to continue.

12. You are prompted to configure the password vault. Enter the Vault keystore password in the appropriate fields and select **Next**.

13. You are presented with a screen specifying default database settings. The installer creates a database that is used to hold data for various components of FSW, including BPEL, jBPM and S-RAMP.

    Change the username and password to something secure and select **Next** to continue.

WARNING

The default database is an **H2** database. Red Hat recommends configuring an alternative database for production systems, using the advanced configuration options.

14. You are presented with a list of packages that will be installed. Select **Next** to proceed.

15. The installation will commence. A status bar for each component will display its progress. Once this is complete, select **Next**.

16. Additional processing of tasks will commence. When the status bar indicates this has been completed, select **Next**.

17. The console displays a message to let you know that the application has been successfully installed. Also, it offers you a list of URLs for accessing the Administration and BPEL Consoles, S-RAMP and DTGov User Interfaces and Gadget Server.

    If you want to repeat the same installation on other machines, select **Generate installation script and properties file** and choose a location and filename for the script.

18. Select **Done**.

Result

The basic installation is complete.

NOTE

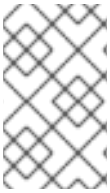
When running the installer on Windows, you may be prompted to provide administrator credentials during installation. To prevent this from happening, add the **izpack.mode=privileged** option to the installation command:

```
java -Dizpack.mode=privileged -jar jboss-fsw-installer-6.0.0.GA-redhat-4.jar
```

Report a bug

CHAPTER 5. STARTING AND STOPPING THE APPLICATION SERVER

You need to start the JBoss Enterprise Application Platform instance for JBoss Fuse Service Works to run. This is because the JBoss Fuse Service Works components run on the JBoss Enterprise Application Platform container.



NOTE

For more information about starting and stopping JBoss Enterprise Application Platform using alternative and more advanced methods, see the *Red Hat JBoss Enterprise Application Platform Administration and Configuration Guide.*

Report a bug

5.1. START JBOSS EAP 6 AS A STANDALONE SERVER

Summary

This topic covers the steps to start JBoss EAP 6 as a Standalone Server.

Procedure 5.1. Start the Platform Service as a Standalone Server

1. **For Red Hat Enterprise Linux.**
   Run the command: ***EAP_HOME*`/bin/standalone.sh`**

2. **For Microsoft Windows Server.**
   Run the command: ***EAP_HOME*`\bin\standalone.bat`**

3. **Optional: Specify additional parameters.**
   To print a list of additional parameters to pass to the start-up scripts, use the ***-h*** parameter.

Result

The JBoss EAP 6 Standalone Server instance starts.

Report a bug

5.2. STOP JBOSS EAP 6 AS A STANDALONE SERVER

You can stop JBoss Enterprise Application Platform using the Management CLI or by pressing **`CTRL+C`** in the terminal.

1. **Stopping JBoss Enterprise Application Platform using the Management CLI.**
   - Run the ***EAP_HOME*`/bin/jboss-cli.sh`** command to launch the Management CLI.

     ```
     $ EAP_HOME/bin/jboss-cli.sh
     ```

   - Run the **`connect`** command to connect to the server.

     ```
     [disconnected /] connect
     ```

- Run the **shutdown** command to stop the server.

```
[standalone@localhost:9999 /] shutdown
```

- Run the **quit** command to close the Management CLI.

```
[standalone@localhost:9999 /] quit
```

2. **Stopping JBoss Enterprise Application Platform by pressing CTRL+C.**
   Press **Ctrl+C** in the *server window* (the terminal window where JBoss Enterprise Application Platform was started).

   Result

   Each of these alternatives stops JBoss Enterprise Application Platform.

Report a bug

CHAPTER 6. MAVEN REPOSITORIES

6.1. ABOUT MAVEN

Apache Maven is a distributed build automation tool used in Java application development to build and manage software projects. Maven uses configuration XML files called POM (Project Object Model) to define project properties and manage the build process. POM files describe the project's module and component dependencies, build order, and targets for the resulting project packaging and output. This ensures that projects are built in a correct and uniform manner.

Maven uses repositories to store Java libraries, plug-ins, and other build artifacts. Repositories can be either local or remote. A local repository is a download of artifacts from a remote repository cached on a local machine. A remote repository is any other repository accessed using common protocols, such as **`http://`** when located on an HTTP server, or **`file://`** when located on a file server. The default repository is the public remote Maven 2 Central Repository.

Configuration of Maven is performed by modifying the **`settings.xml`** file. You can either configure global Maven settings in the ***`M2_HOME`*`/conf/settings.xml`** file, or user-level settings in the ***`USER_HOME`*`/.m2/settings.xml`** file.

For more information about Maven, see Welcome to Apache Maven.

For more information about Maven repositories, see Apache Maven Project - Introduction to Repositories.

For more information about Maven POM files, see the Apache Maven Project POM Reference.

NOTE

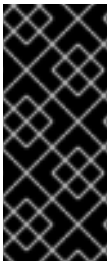
Red Hat JBoss Fuse Service Works has been built with maven 3.0.x Therefore, this is the recommended maven version for building your own SwitchYard applications.

Report a bug

6.2. ABOUT THE PROVIDED MAVEN REPOSITORIES

A set of repositories containing artifacts required to build applications based on Red Hat JBoss Fuse Service Works is provided with this release. Maven must be configured to use these repositories and the Maven Central Repository in order to provide correct build functionality.

Two interchangeable sets of repositories ensuring the same functionality are provided. The first set is available for download and storage in a local file system, the second set is hosted online for use as remote repositories. If you provided the location of Maven's **`settings.xml`** file during installation, Maven is already configured to use the online repositories. If you did not provide the location during installation, you need to configure Maven manually by following the procedure in Section 6.4, “Configuring Maven to Use the Online Repositories” or Section 6.3, “Configuring Maven to Use the File System Repositories”.



IMPORTANT

The set of online remote repositories is a technology preview source of components. As such, it is not in scope of patching and is supported only for use in development environment. Using the set of online repositories in production environment is a potential source of security vulnerabilities and is therefore not a supported use case. For more information see https://access.redhat.com/site/maven-repository.

Report a bug

6.3. CONFIGURING MAVEN TO USE THE FILE SYSTEM REPOSITORIES

Overview

In situations where you cannot use the online repositories, you will have to download and configure the required repositories locally.

Procedure 6.1.

1. Download the following ZIP archives containing the required repositories:
   - http://maven.repository.redhat.com/techpreview/fsw6/6.0.0/fsw-6.0.0.GA-redhat-2-repository.zip
   - http://maven.repository.redhat.com/techpreview/eap6/6.1.1/jboss-eap-6.1.1-maven-repository.zip

   NOTE

   Alternatively, the ZIP archives can be downloaded from http://access.redhat.com/jbossnetwork/.

2. Unzip the downloaded ZIP files into an arbitrary location in a local file system.
3. Add entries for the unzipped repositories to Maven's **`settings.xml`** file. The following code sample contains a profile with the repositories and an activation entry for the profile:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/xsd/settings-1.0.0.xsd">
  <localRepository/>
  <profiles>
    <!-- Profile with local repositories required by Fuse Service
Works -->
    <profile>
      <id>fsw-local-repos</id>
      <repositories>
        <repository>
          <id>fsw-6.0.0.GA-redhat-2-repository</id>
          <name>FSW 6.0.0.GA Repository</name>
          <url>file://<!-- path to the repository -->/fsw-6.0.0.GA-
redhat-2-repository</url>
          <layout>default</layout>
```

```
          <releases>
            <enabled>true</enabled>
            <updatePolicy>never</updatePolicy>
          </releases>
          <snapshots>
            <enabled>false</enabled>
            <updatePolicy>never</updatePolicy>
          </snapshots>
        </repository>
        <repository>
          <id>jboss-eap-6.1.1.GA-maven-repository</id>
          <name>EAP 6.1.1.GA Repository</name>
          <url>file://<!-- path to the repository -->/jboss-eap-
6.1.1.GA-maven-repository</url>
          <layout>default</layout>
          <releases>
            <enabled>true</enabled>
            <updatePolicy>never</updatePolicy>
          </releases>
          <snapshots>
            <enabled>false</enabled>
            <updatePolicy>never</updatePolicy>
          </snapshots>
        </repository>
      </repositories>
      <pluginRepositories>
        <pluginRepository>
          <id>fsw-6.0.0.GA-redhat-2-repository</id>
          <name>FSW 6.0.0.GA Repository</name>
          <url>file://<!-- path to the repository -->/fsw-6.0.0.GA-
redhat-2-repository</url>
          <layout>default</layout>
          <releases>
            <enabled>true</enabled>
            <updatePolicy>never</updatePolicy>
          </releases>
          <snapshots>
            <enabled>false</enabled>
            <updatePolicy>never</updatePolicy>
          </snapshots>
        </pluginRepository>
        <pluginRepository>
          <id>jboss-eap-6.1.1.GA-maven-repository</id>
          <name>EAP 6.1.1 GA Repository</name>
          <url>file://<!-- path to the repository -->/jboss-eap-
6.1.1.GA-maven-repository</url>
          <layout>default</layout>
          <releases>
            <enabled>true</enabled>
            <updatePolicy>never</updatePolicy>
          </releases>
          <snapshots>
            <enabled>false</enabled>
            <updatePolicy>never</updatePolicy>
          </snapshots>
        </pluginRepository>
```

```
      </pluginRepositories>
    </profile>
  </profiles>
  <activeProfiles>
   <!-- Activation of the Fuse Service Works profile -->
   <activeProfile>fsw-local-repos</activeProfile>
  </activeProfiles>
</settings>
```

IMPORTANT

The **bpm-service** quickstart application will not build without two additional repositories. If you want to build this application, download the following repositories and add them to your **setting.xml** file in the same way as that shown above.

- http://maven.repository.redhat.com/techpreview/eap6/6.0.1/jboss-eap-6.0.1-maven-repository.zip
- http://maven.repository.redhat.com/techpreview/wfk2/2.5.0/jboss-wfk-2.5.0-maven-repository.zip

Result

The Maven repositories are downloaded, unzipped in a local file system, registered in Maven's **settings.xml** file, and ready to be used when performing Maven builds.

Troubleshooting

Q: Why does the bpm-service quickstart application fail on build?

A: Issue

When you build or deploy the **bpm-service** quickstart application, it fails with the following error:

```
[ERROR] Failed to execute goal on project switchyard-quickstart-bpm-
service: Could not resolve dependencies for project
org.switchyard.quickstarts:switchyard-quickstart-bpm-
service:jar:1.1.1-p5-redhat-1: The following artifacts could not be
resolved: org.jboss.netty:netty:jar:3.2.6.Final-redhat-2,
org.apache.lucene:lucene-queryparser:jar:3.6.2-redhat-4: Could not
find artifact org.jboss.netty:netty:jar:3.2.6.Final-redhat-2 in fsw-
6.0.0.GA-redhat-2-repository (file:///home/belong/maven_repos/fsw-
6.0.0.GA-redhat-2-repository) -> [Help 1]
```

Cause

The **bpm-service** quickstart application requires an additional two dependencies, provided by the JBoss EAP 6.0.1 and JBoss Web Framework Kit 2.5.0 Maven repositories.

Resolution

1. Download the following ZIP archives, and unzip them into a location of your choice:

   http://maven.repository.redhat.com/techpreview/eap6/6.0.1/jboss-eap-6.0.1-maven-repository.zip

http://maven.repository.redhat.com/techpreview/wfk2/2.5.0/jboss-wfk-2.5.0-maven-repository.zip

2. Add entries for the unzipped repositories to your Maven **`settings.xml`** file:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/xsd/settings-
1.0.0.xsd">
  <localRepository/>
  <profiles>
    <!-- Profile with local repositories required by Fuse
Service Works -->
    <profile>
      <id>fsw-local-repos</id>
      <repositories>
        <repository>
          <id>fsw-6.0.0.GA-redhat-2-repository</id>
          <name>FSW 6.0.0.GA Repository</name>
          <url>file://<!-- path to the repository -->/fsw-
6.0.0.GA-redhat-2-repository</url>
          <layout>default</layout>
          <releases>
            <enabled>true</enabled>
            <updatePolicy>never</updatePolicy>
          </releases>
          <snapshots>
            <enabled>false</enabled>
            <updatePolicy>never</updatePolicy>
          </snapshots>
        </repository>
        <repository>
          <id>jboss-eap-6.1.1.GA-maven-repository</id>
          <name>EAP 6.1.1.GA Repository</name>
          <url>file://<!-- path to the repository -->/jboss-
eap-6.1.1.GA-maven-repository</url>
          <layout>default</layout>
          <releases>
            <enabled>true</enabled>
            <updatePolicy>never</updatePolicy>
          </releases>
          <snapshots>
            <enabled>false</enabled>
            <updatePolicy>never</updatePolicy>
          </snapshots>
        </repository>
        <repository>
          <id>jboss-wfk-2.5.0-maven-repository</id>
          <name>JBoss WFK 2.5.0 Repository</name>
          <url>file://<!-- path to the repository -->/jboss-
wfk-2.5.0-maven-repository</url>
          <layout>default</layout>
          <releases>
            <enabled>true</enabled>
            <updatePolicy>never</updatePolicy>
```

```
          </releases>
          <snapshots>
            <enabled>false</enabled>
            <updatePolicy>never</updatePolicy>
          </snapshots>
        </repository>
        <repository>
          <id>jboss-eap-6.0.1-maven-repository</id>
          <name>EAP 6.0.1 Repository</name>
          <url>file://<!-- path to the repository -->/jboss-
eap-6.0.1-maven-repository</url>
          <layout>default</layout>
          <releases>
            <enabled>true</enabled>
            <updatePolicy>never</updatePolicy>
          </releases>
          <snapshots>
            <enabled>false</enabled>
            <updatePolicy>never</updatePolicy>
          </snapshots>
        </repository>
      </repositories>
      <pluginRepositories>
        <pluginRepository>
          <id>fsw-6.0.0.GA-redhat-2-repository</id>
          <name>FSW 6.0.0.GA Repository</name>
          <url>file://<!-- path to the repository -->/fsw-
6.0.0.GA-redhat-2-repository</url>
          <layout>default</layout>
          <releases>
            <enabled>true</enabled>
            <updatePolicy>never</updatePolicy>
          </releases>
          <snapshots>
            <enabled>false</enabled>
            <updatePolicy>never</updatePolicy>
          </snapshots>
        </pluginRepository>
        <pluginRepository>
          <id>jboss-eap-6.1.1.GA-maven-repository</id>
          <name>EAP 6.1.1 GA Repository</name>
          <url>file://<!-- path to the repository -->/jboss-
eap-6.1.1.GA-maven-repository</url>
          <layout>default</layout>
          <releases>
            <enabled>true</enabled>
            <updatePolicy>never</updatePolicy>
          </releases>
          <snapshots>
            <enabled>false</enabled>
            <updatePolicy>never</updatePolicy>
          </snapshots>
        </pluginRepository>
        <pluginRepository>
          <id>jboss-wfk-2.5.0-maven-repository</id>
          <name>JBoss WFK 2.5.0 Repository</name>
```

```
          <url>file://<!-- path to the repository -->/jboss-
wfk-2.5.0-maven-repository</url>
          <layout>default</layout>
          <releases>
            <enabled>true</enabled>
            <updatePolicy>never</updatePolicy>
          </releases>
          <snapshots>
            <enabled>false</enabled>
            <updatePolicy>never</updatePolicy>
          </snapshots>
        </pluginRepository>
        <pluginRepository>
          <id>jboss-eap-6.0.1-maven-repository</id>
          <name>EAP 6.0.1 Repository</name>
          <url>file://<!-- path to the repository -->/jboss-
eap-6.0.1-maven-repository</url>
          <layout>default</layout>
          <releases>
            <enabled>true</enabled>
            <updatePolicy>never</updatePolicy>
          </releases>
          <snapshots>
            <enabled>false</enabled>
            <updatePolicy>never</updatePolicy>
          </snapshots>
        </pluginRepository>
      </pluginRepositories>
    </profile>
  </profiles>
  <activeProfiles>
   <!-- Activation of the Fuse Service Works profile -->
   <activeProfile>fsw-local-repos</activeProfile>
  </activeProfiles>
</settings>
```

Q: Why do I still get errors when building or deploying my applications?

A: Issue

When you build or deploy a project, it fails with one or both of the following errors:

```
[ERROR] Failed to execute goal on project PROJECT_NAME

Could not find artifact ARTIFACT_NAME
```

Cause

Your cached local Maven repository might contain outdated artifacts.

Resolution

To resolve the issue, delete the cached local repository – the **~/.m2/repository/** directory on Linux or the ***%SystemDrive%*\Users*USERNAME*\.m2\repository** directory on Windows – and run **mvn clean install -U**. This will force Maven to download correct versions of required artifacts when performing the next build.

Q: Why is JBoss Developer Studio using my old Maven configuration?

A: Issue

You have updated your Maven configuration, but this configuration is not reflected in JBoss Developer Studio.

Cause

If JBoss Developer Studio is running when you modify your Maven **`settings.xml`** file, this configuration will not be reflected in JBoss Developer Studio.

Resolution

Refresh the Maven settings in the IDE. From the menu, choose **Window → Preferences**. In the **`Preferences`** Window, expand **`Maven`** and choose **`User Settings`**. Click the **`Update Settings`** button to refresh the Maven user settings in JBoss Developer Studio.

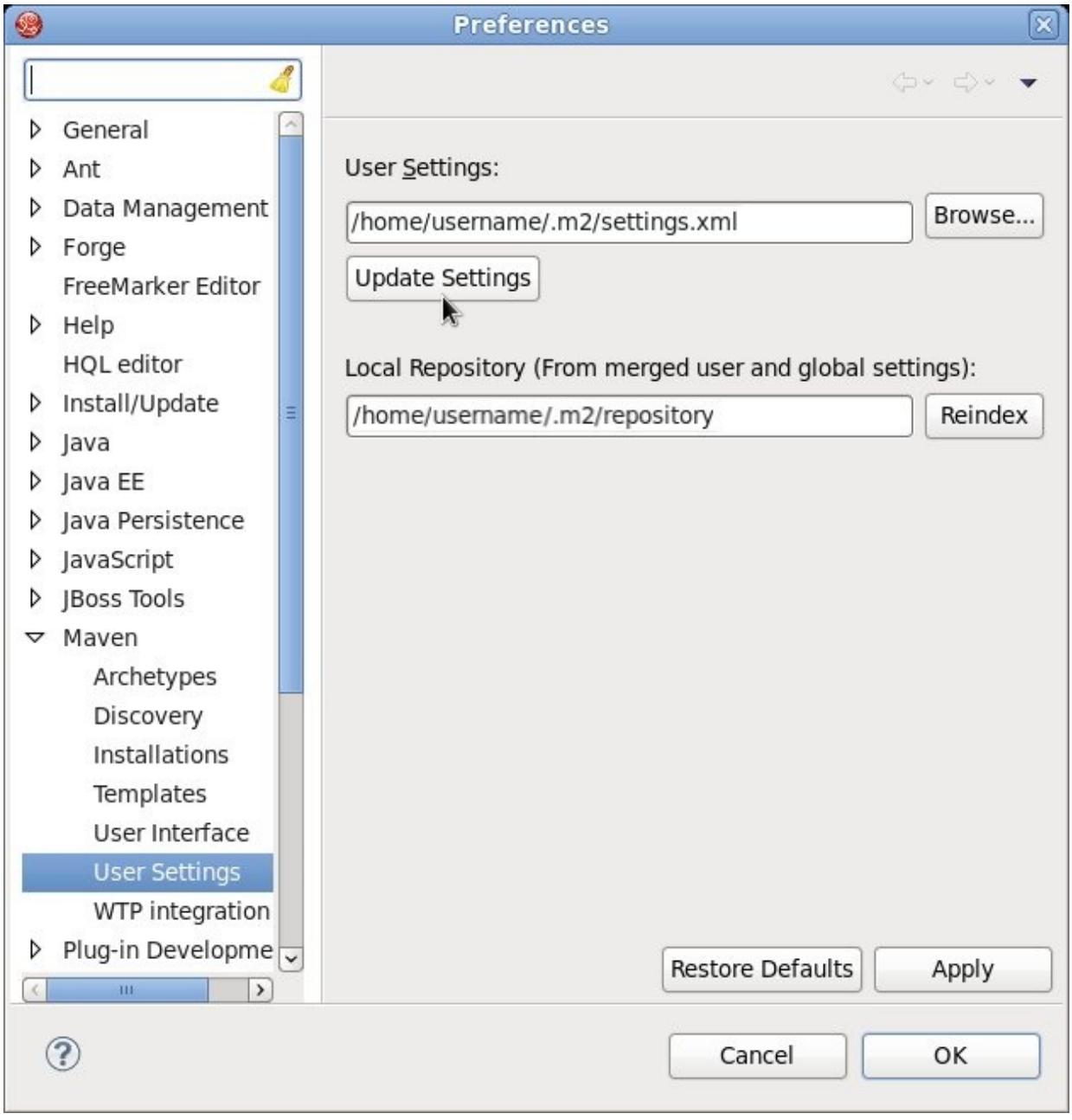


Figure 6.1. Update Maven User Settings

Report a bug

6.4. CONFIGURING MAVEN TO USE THE ONLINE REPOSITORIES

The online repositories required for Red Hat JBoss Fuse Service Works applications are located at http://maven.repository.redhat.com/techpreview/all/ and http://repository.jboss.org/nexus/content/repositories/public/.

If you provided the location of Maven's **settings.xml** file during installation, Maven is already configured to use the online repositories. If you did not provide the location during installation, you need to configure Maven manually by following the procedure in Section 6.4, “Configuring Maven to Use the Online Repositories” or Section 6.3, “Configuring Maven to Use the File System Repositories”.

If you did not configure the Maven repository during installation, you can configure it using the following procedure. (It is also possible to do this using the project's POM file, but this is not recommended.)

Procedure 6.2. Configuring Maven to Use the Online Repositories

1. Add entries for the online repositories to Maven's **settings.xml** file as in the code sample below:

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
http://maven.apache.org/xsd/settings-1.0.0.xsd">

  <profiles>
    <!-- Profile with online repositories required by Fuse Service
Works -->
    <profile>
      <id>fsw-online-repos</id>
      <repositories>
        <repository>
          <id>jboss-ga-repository</id>

<url>http://maven.repository.redhat.com/techpreview/all</url>
          <releases>
            <enabled>true</enabled>
          </releases>
          <snapshots>
            <enabled>false</enabled>
          </snapshots>
        </repository>
        <repository>
          <id>jboss-public-repository</id>

<url>http://repository.jboss.org/nexus/content/repositories/public/<
/url>
          <releases>
            <enabled>true</enabled>
          </releases>
          <snapshots>
            <enabled>false</enabled>
```

```
          </snapshots>
        </repository>
      </repositories>
      <pluginRepositories>
        <pluginRepository>
          <id>jboss-ga-plugin-repository</id>

<url>http://maven.repository.redhat.com/techpreview/all</url>
          <releases>
            <enabled>true</enabled>
          </releases>
          <snapshots>
            <enabled>false</enabled>
          </snapshots>
        </pluginRepository>
        <pluginRepository>
          <id>jboss-public-plugin-repository</id>

<url>http://repository.jboss.org/nexus/content/repositories/public/<
/url>
          <releases>
            <enabled>true</enabled>
          </releases>
          <snapshots>
            <enabled>false</enabled>
          </snapshots>
        </pluginRepository>
      </pluginRepositories>
    </profile>
  </profiles>

  <activeProfiles>
    <!-- Activation of the Fuse Service Works profile -->
    <activeProfile>fsw-online-repos</activeProfile>
  </activeProfiles>

</settings>
```

2. If you modified the **`settings.xml`** file while JBoss Developer Studio was running, you must refresh Maven settings in the IDE. From the menu, choose **Window** → **Preferences**. In the **`Preferences`** Window, expand **`Maven`** and choose **`User Settings`**. Click the **`Update Settings`** button to refresh the Maven user settings in JBoss Developer Studio.

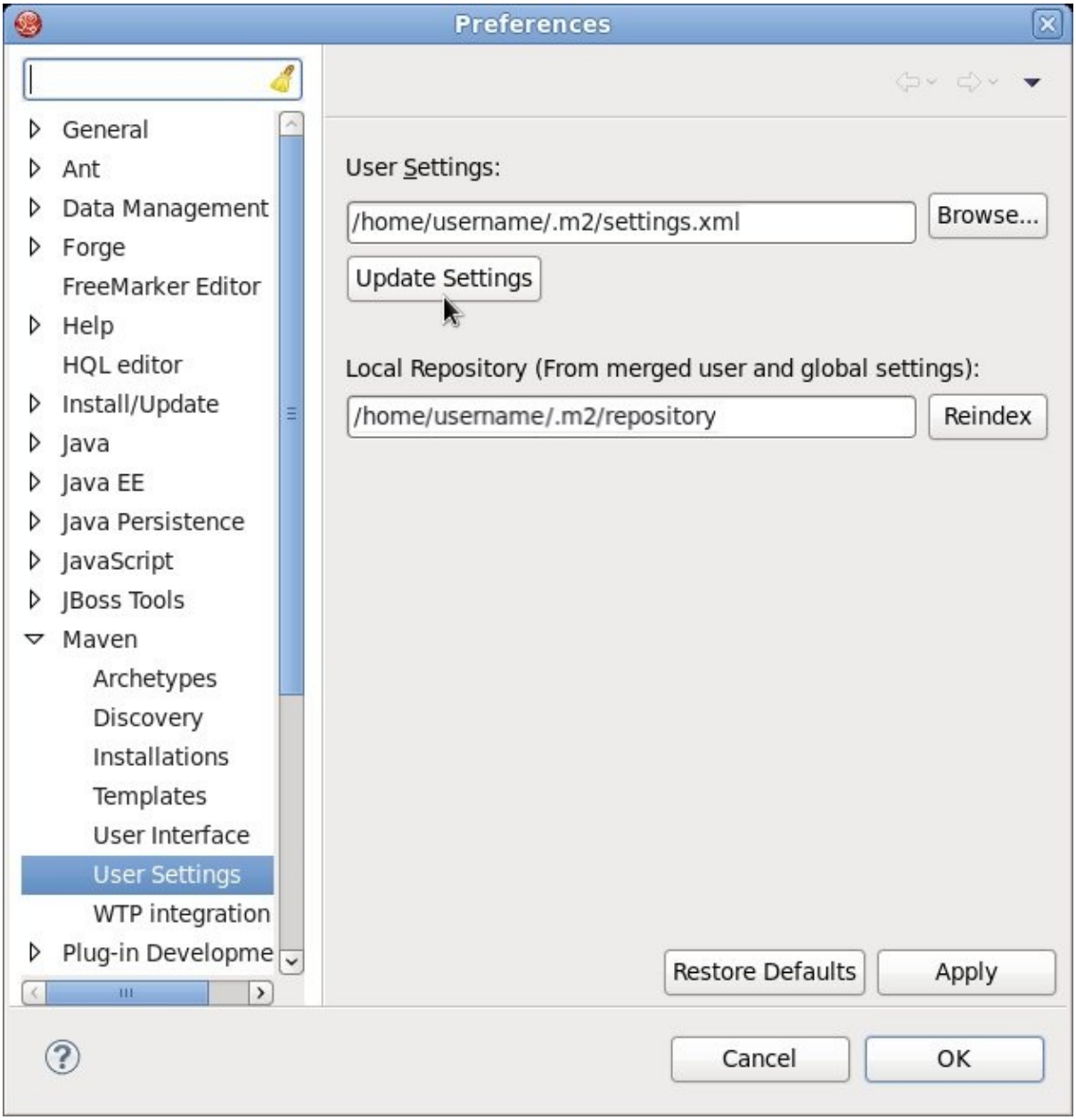


Figure 6.2. Update Maven User Settings

Result

Maven has been configured to use the online repositories provided for Red Hat JBoss Fuse Service Works.

IMPORTANT

If your cached local Maven repository contains outdated artifacts, you may encounter one of the following Maven errors when you build or deploy your project:

- Missing artifact *ARTIFACT_NAME*
- [ERROR] Failed to execute goal on project *PROJECT_NAME*; Could not resolve dependencies for *PROJECT_NAME*

To resolve the issue, delete the cached local repository – the **~/.m2/repository/** directory on Linux or the ***%SystemDrive%*\Users*USERNAME*\.m2\repository** directory on Windows. This will force Maven to download correct versions of required artifacts during the next build.

Report a bug

6.5. DEPENDENCY MANAGEMENT

In order to use correct Maven dependencies in your Red Hat JBoss Fuse Service Works project, relevant Bill Of Materials (BOM) and parent POM files must be added to the project's **pom.xml** file. Adding the BOM and parent will ensure that correct versions of plugins and transitive dependencies from the provided Maven repositories are included in the project.

To ensure correct dependency usage in your project, declare the following parent in the project's **pom.xml** file:

- **org.jboss.ip.component.management:ip-parent:1.1-redhat-5**

and add the following two BOM files as dependencies in the **dependencyManagement** section:

- **org.jboss.ip.component.management:ip-dependency-management-all:1.1-redhat-5**
- **org.jboss.component.management:jboss-dependency-management-all:6.1.1.Final-redhat-61**

Use the entries from the code sample below for this purpose.

```
<parent>
  <groupId>org.jboss.ip.component.management</groupId>
  <artifactId>ip-parent</artifactId>
  <version>1.1-redhat-5</version>
</parent>

...

<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.jboss.ip.component.management</groupId>
      <artifactId>ip-dependency-management-all</artifactId>
      <type>pom</type>
      <version>1.1-redhat-5</version>
      <scope>import</scope>
```

```
    </dependency>
    <dependency>
      <groupId>org.jboss.component.management</groupId>
      <artifactId>jboss-dependency-management-all</artifactId>
      <type>pom</type>
      <version>6.1.1.Final-redhat-61</version>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

Report a bug

CHAPTER 7. INSTALLING JBOSS DEVELOPER STUDIO

7.1. INSTALL JBOSS DEVELOPER STUDIO

Procedure 7.1. Install with the Universal Installer

1. Log into the Customer Portal at https://access.redhat.com.
2. From the menu bar click **Downloads**.
3. Under **Red Hat JBoss Middleware**, click **Download Software**.

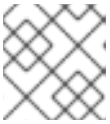
Figure 7.1. Download Red Hat JBoss Middleware Software on the Customer Portal

4. Under **`Software Downloads`**, select the following options:
   - From the **`Product`** list, select **`JBoss Developer Studio`**.
   - From the **`Version`** list, select **`7.1.x`**.
5. - For the JBoss Developer Studio universal installer, click **`Download`** for the **`Red Hat JBoss Developer Studio 7.1.x Stand Alone Universal Binary`** download file.
   - For the JBoss Developer Studio and JBoss EAP universal installer, click **`Download`** for the **`Red Hat JBoss Developer Studio 7.1.x Universal Binary with JBoss EAP`** download file.

   This downloads a universal installer **`.jar`** file.
6. On the command line, navigate to **`path/to/.jar`** and enter

```
java -jar jbdevstudio-product-universal-version.jar
```

where *version* is substituted to match the name of the **`.jar`** file.



NOTE

Alternatively, to start the installer you may be able to double-click the **`.jar`** file.

7. When the **Installer** window opens, click **Next**.

8. After reading and agreeing to the terms of the End User License Agreement, click **I accept the terms of this license agreement** and click **Next**.

9. In the **Select the installation path** field, type the path where you want JBoss Developer Studio to be installed or click **Browse** to navigate to the location. When the **Select the installation path** field shows the correct path, click **Next**. When you are prompted about the specified location being created or overwritten, review the message and, if satisfied, click **OK** or **Yes** as appropriate.

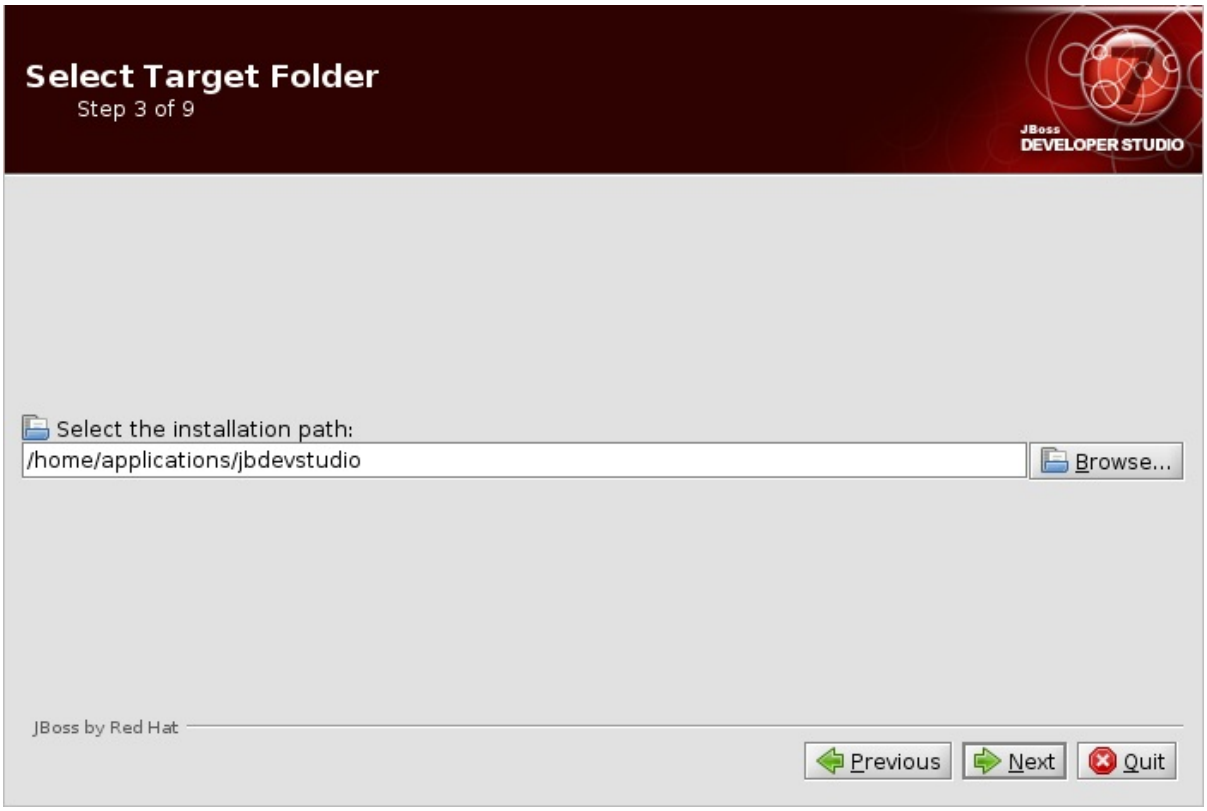


Figure 7.2. Installation Step 3: Select Target Folder

10. In the **Select Java VM** step, **Default Java VM** is automatically selected. Ensure that the disabled text field contains the path of the Java developer kit you want to use. This is based on the default Java developer kit of your system. To change the specified Java developer kit, click **Specific Java VM** and type the path of the Java developer kit in the text field or use the **Browse** button to locate the Java developer kit. When the text field shows the correct Java developer kit path, click **Next**.

IMPORTANT

You must specify a Java developer kit with a 32-bit JRE to install a 32-bit version of JBoss Developer Studio and a 64-bit JRE to install a 64-bit version of JBoss Developer Studio. To change the bit version of the Java developer kit to be used for installing JBoss Developer Studio, complete the appropriate step for your operating system:

- On OS X operating systems, from the **`Installation type`** list click the appropriate bit version.
- On Linux distributions and Microsoft Windows operating systems, in the text field type the path to the appropriate bit version of the Java developer kit.

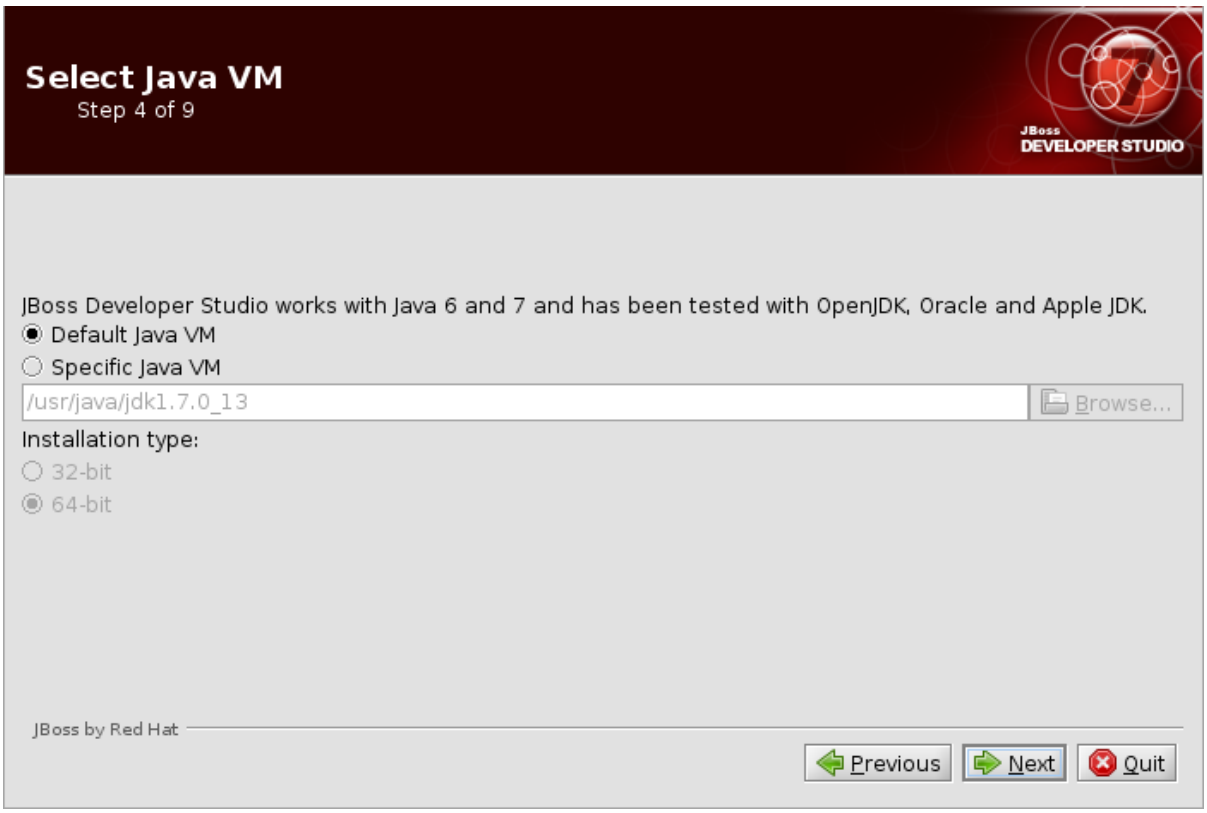


Figure 7.3. Installation Step 4: Select Java VM

11. Review the details in the **Summary Information** window and, if they are correct, click **`Next`**. Installation commences.
12. When the **`Pack installation progress`** bar shows **`Finished`**, click **`Next`**. The installation process is now complete.

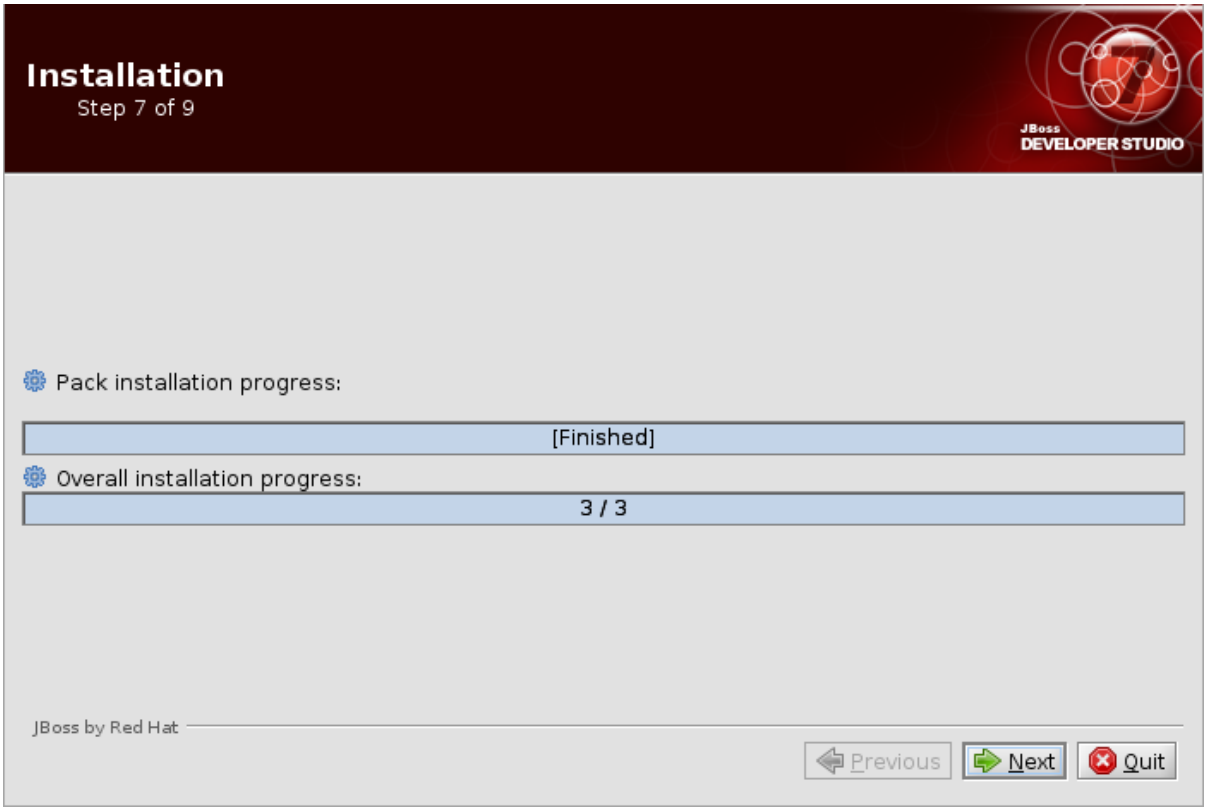
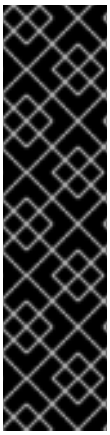


Figure 7.4. Installation Step 7: Installation Progress in Finished State

13. To create shortcuts for starting JBoss Developer Studio, select the **Create shortcuts in the Start-Menu** and **Create additional shortcut on the desktop** check boxes and click **Next**.

14. To automatically start JBoss Developer Studio when the **Installer** window closes, select the **Run JBoss Developer Studio after installation** check box. Click **Done** to close the **Installer** window.



IMPORTANT

Linux distributions have a maximum number of files that a process can have open at one time. If this maximum number of files is set too low, JBoss Developer Studio will not start. You must open the **/etc/security/limits.conf** file and ensure that the **soft nofile** and **hard nofile** variables have values of **9216** at a minimum. If the variables have smaller values, the values must be increased to **9216**. If the variables are not specified, the following lines must be added to the file:

```
* soft nofile 9216
* hard nofile 9216
```

Report a bug

7.2. INSTALLING JBOSS DEVELOPER STUDIO INTEGRATION STACK

JBoss Developer Studio Integration Stack is not packaged as part of JBoss Developer Studio installations. These plug-ins must be installed independently through JBoss Central, as detailed in the procedure below.

Procedure 7.2. Install JBoss Developer Studio Integration Stack

1. Start JBoss Developer Studio.

2. In JBoss Central, select the **Software/Update** tab. Scroll through the list to locate **JBoss Developer Studio Integration Stack**. Select the check box next to **JBoss Integration and SOA Development** and click **Install**.

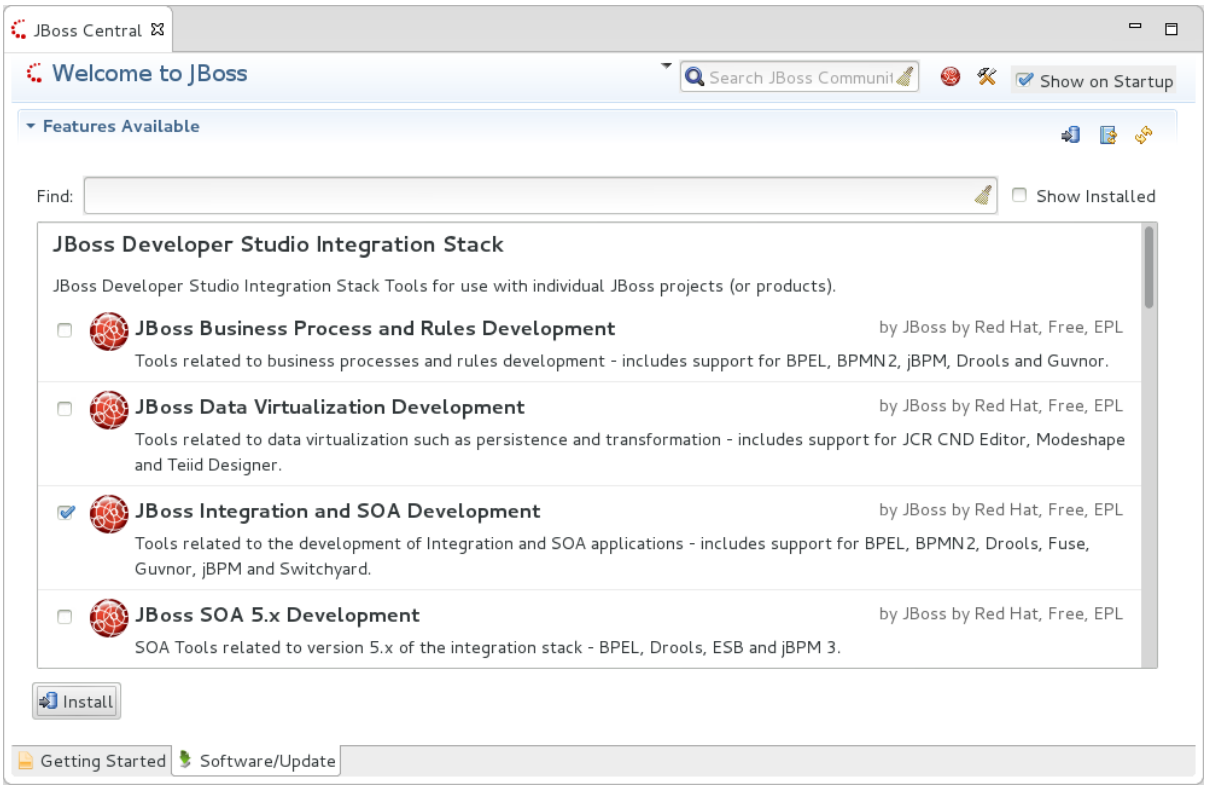


Figure 7.5. Find JBoss Developer Studio Integration Stack in JBoss Central Software/Update Tab

3. In the **Install** wizard, ensure the check boxes are selected for the software you want to install and click **Next**. It is recommended that you install all of the selected components.

4. Review the details of the items listed for install and click **Next**. After reading and agreeing to the license(s), click **I accept the terms of the license agreement(s)** and click **Finish**. The **Installing Software** window opens and reports the progress of the installation.

5. During the installation process you may receive warnings about installing unsigned content. If this is the case, check the details of the content and if satisfied click **OK** to continue with the installation.

Figure 7.6. Warning Prompt for Installing Unsigned Content

6. Once installing is complete, you are prompted to restart the IDE. Click **Yes** to restart now and **No** if you need to save any unsaved changes to open projects. Note that changes do not take effect until the IDE is restarted.

Once installed, you may need to complete additional configuration actions before you can use the individual JBoss Developer Studio Integration Stack components. For plug-in specific configuration information, see the appropriate Red Hat JBoss product documentation available from https://access.redhat.com/site/documentation on the Red Hat Customer Portal.

IMPORTANT

The installation method for early releases of JBoss Developer Studio Integration Stack may vary from that given here. For instructions, see the appropriate Red Hat JBoss product documentation available from https://access.redhat.com/site/documentation on the Red Hat Customer Portal.

Report a bug

CHAPTER 8. GETTING STARTED WITH SWITCHYARD

8.1. THE REMOTE INVOKER QUICKSTART APPLICATION

8.1.1. Overview of the Remote Invoker Quickstart

The Remote Invoker serves as a remote invocation client for SwitchYard services. It allows non-SwitchYard applications to invoke any service in SwitchYard which uses a <binding.sca> binding.

This section includes the use-case of Remote Invoker quickstart. It demonstrates how to use the Remote Invoker with services in SwitchYard using the SCA Binding.

Remote Invoker

To evaluate an offer on an automobile and make a decision on whether to accept the offer or not. You need to implement the CreditCheck service using business rules and the Dealer service using CDI. The Dealer service evaluates the offer and submits the applicant to a credit check, before replying with an answer to the client.

This process includes unit tests to test each service individually and a test driver called RemoteClient which demonstrates the use of an HTTP-based Remote Invoker to invoke a SwitchYard service remotely.

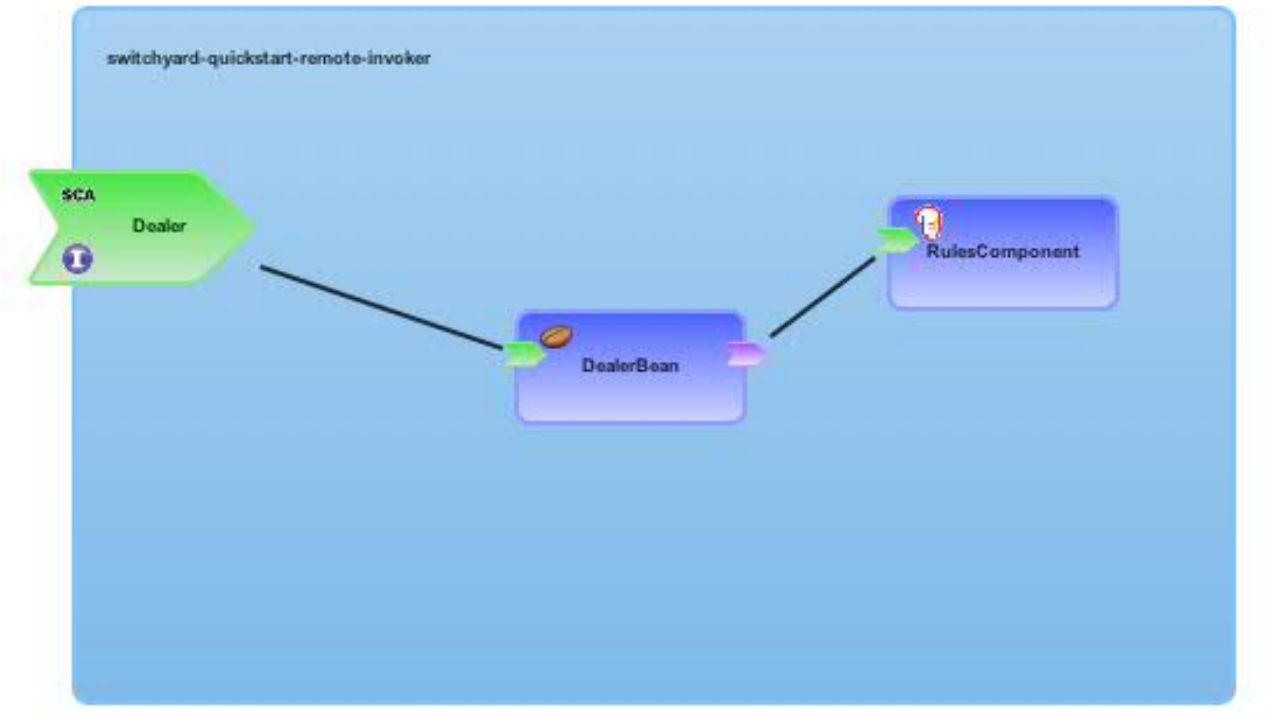


Figure 8.1. Remote Invoker Image

Report a bug

8.1.2. Using the Remote Invoker Quickstart

Prerequisites

- Maven 3.0.x must be installed.

Procedure 8.1. Task

1. Open a terminal.
2. Navigate to the **`EAP_HOME/quickstarts/switchyard/remote-invoker`** directory.
3. Run the following command:

```
mvn clean install
```

Result

The Remote Invoker quickstart is ready for use.

Report a bug

8.1.3. Running the Remote Invoker Quickstart Application

Overview

This topic demonstrates how to run and test the Remote Invoker quickstart application on the command line using Maven.

Prerequisites

The application server must be running.

Procedure 8.2. Task

1. Navigate to the Remote Invoker quickstart application directory:

```
cd EAP_HOME/quickstarts/switchyard/remote-invoker
```

2. Run the following command to deploy the quickstart application:

```
mvn jboss-as:deploy
```

This will package the Remote Invoker quickstart in ***`EAP_HOME`*`/quickstarts/switchyard/remote-invoker/target/switchyard-quickstart-remote-invoker.jar`** and deploy it to the application server. You can confirm it has been deployed by checking the application server log or by logging into the Management Console and navigating to **Runtime Operations** → **SwitchYard** from the **Runtime** view.

The application is enabled by default.

3. Run the following command to test the quickstart application:

```
mvn exec:java
```

This will run the test class specified by **`mainClass`** in the application's **`pom.xml`** file, in this case, **`EAP_HOME/quickstarts/switchyard/remote-`**

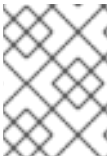
invoker/src/test/java/org/switchyard/quickstarts/remoteinvoker/RemoteClient.java.

You will receive the following message:

```
================================
Was the offer accepted? true
================================
```

You will also see the following message in the server log:

```
Approving credit for John Smith
```



NOTE

You can package a quickstart without deploying it by running **mvn package** from within the quickstart application directory.

See Also:

- Section 5.1, “Start JBoss EAP 6 as a Standalone Server”

Report a bug

8.1.4. Undeploying the Remote Invoker Quickstart Application

Overview

This topic demonstrates how to undeploy the Remote Invoker quickstart application on the command line using Maven.

Prerequisites

The application server must be running.

Procedure 8.3. Task

1. Navigate to the Remote Invoker quickstart application directory:

```
cd EAP_HOME/quickstarts/switchyard/remote-invoker
```

2. Run the following command to remove the deployed quickstart:

```
mvn jboss-as:undeploy
```

Result

The Remote Invoker quickstart is no longer deployed to the application server. You can confirm it has been removed by checking the application server log or by logging into the Management Console and navigating to **Runtime Operations** → **SwitchYard** from the **Runtime** view.

Report a bug

8.2. CREATE A SCHEDULED SERVICE

Sometimes you need a particular service to run periodically, according to a set time or interval.

The **camel-quartz-binding** quickstart application is provided in the ***EAP_HOME*/quickstarts/switchyard/** directory as an example. You can follow the instructions in the provided **Readme.md** file to build and deploy this application.

The remainder of this section describes how to develop your own scheduled service.

Tutorial Overview

By the end of this tutorial you will have the knowledge and skills necessary to develop, deploy and test a scheduled service for your own integrated solutions. The service will be implemented as a SwitchYard application, making use of the **camel-quartz** component. You will develop the application with the JBoss Integration and SOA Development tools, provided as a plug-in for JBoss Developer Studio.

Prerequisites

- The JBoss Integration and SOA Development tools are installed for JBoss Developer Studio.
- Maven is installed and configured.

Procedure

1. **Create a new SwitchYard project in JBoss Developer Studio.**

   a. Select **File** → **New** → **Project**. Then select the appropriate wizard: **SwitchYard** → **SwitchYard Project** and select **Next**.

   b. Choose a project name (we are using **switchyard-example**) and select **Next**.

   c. You are now prompted to provide **SwitchYard Project Configuration**. Choose the **Bean** and **Scheduling** SwitchYard components.

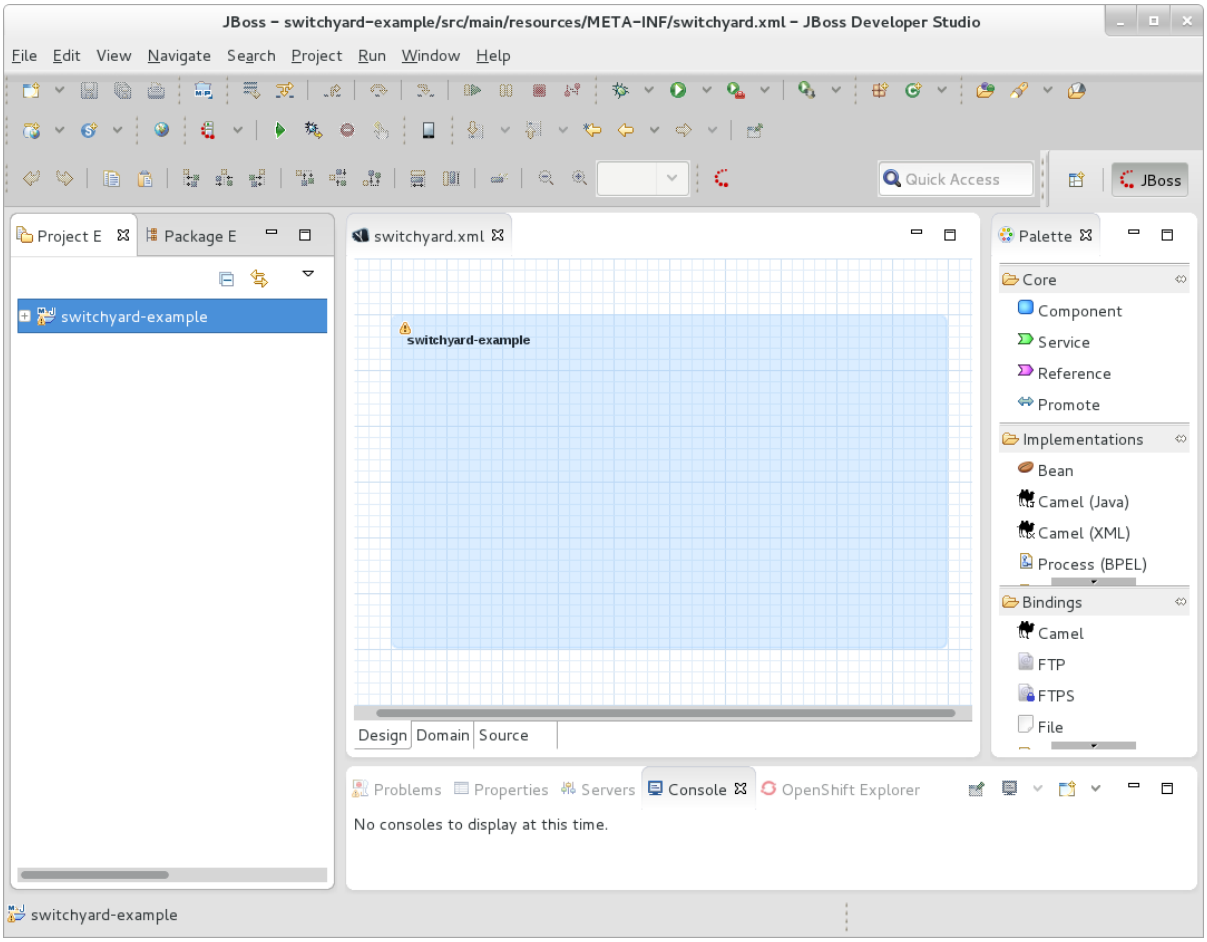
      This automatically adds the following dependencies to the project's **pom.xml** file so they are included when the project is built:

      ```
      <dependency>
        <groupId>org.switchyard.components</groupId>
        <artifactId>switchyard-component-bean</artifactId>
        <version>${switchyard.version}</version>
      </dependency>
      <dependency>
        <groupId>org.switchyard.components</groupId>
        <artifactId>switchyard-component-camel-quartz</artifactId>
        <version>${switchyard.version}</version>
      </dependency>
      ```

   d. Select **Finish** to proceed.

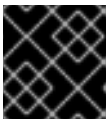
   At this point, the new project is generated and appears in the **Project Explorer** view. You can double-click on the **pom.xml** file to open it, select the **pom.xml** tab, and confirm the dependencies added in the steps above.

Open the **switchyard.xml** file (found under **src/main/resources**). By default, the file opens on the **Design** tab. This large blue canvas is referred to as the *composite*.



Click on the **Source** tab to see the underlying project XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<sy:switchyard ... >
  <sca:composite name="switchyard-example"
targetNamespace="urn:com.example.switchyard:switchyard-example:1.0">
  </sca:composite>
</sy:switchyard>
```



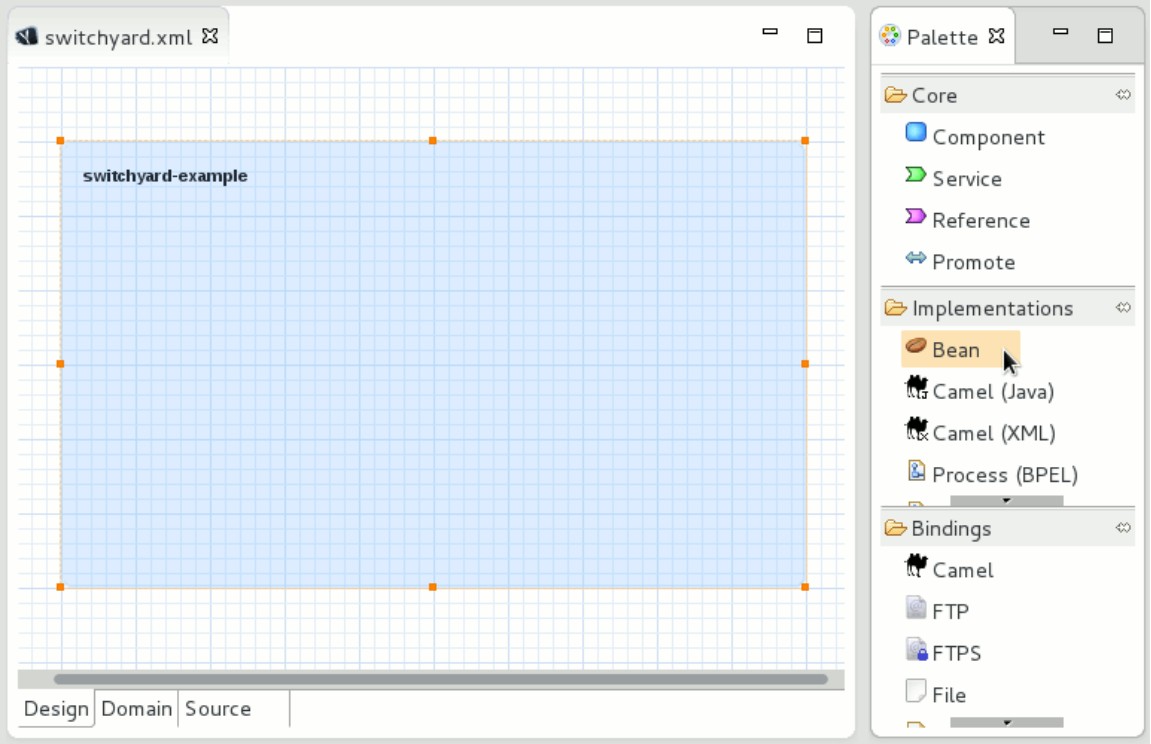
IMPORTANT

Red Hat does not recommend editing this XML directly. Read more...

2. **Add a Bean implementation.**

   a. Click back to the **Design** tab for the **switchyard.xml** file.

   b. Click and drag a **Bean** implementation from the SwitchYard **Palette** onto your composite. This will automatically invoke the **New Bean Service** wizard. If the **Palette** is not yet visible, select **Window** → **Show View** → **Other**, then expand **General** and select **Palette**.

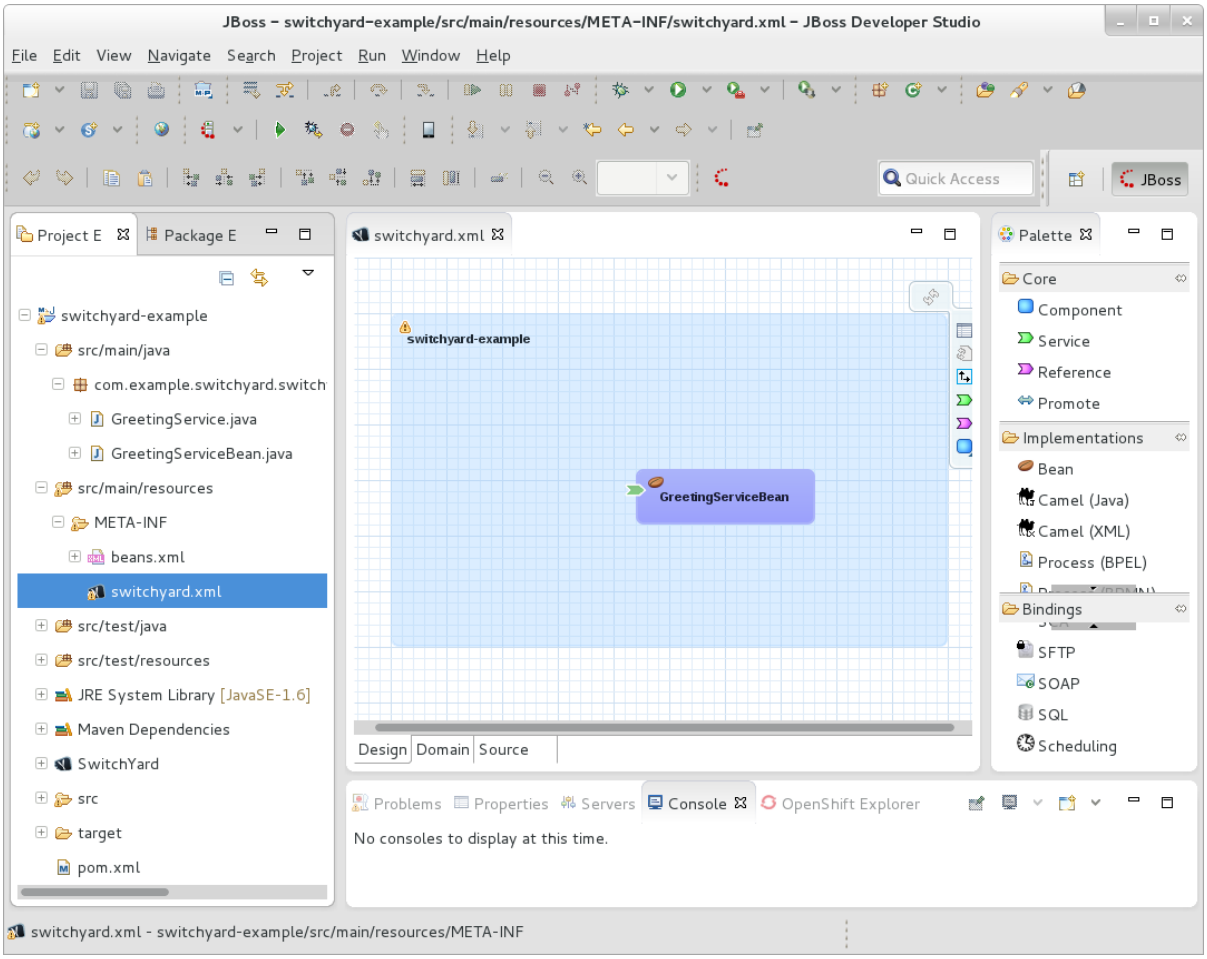


c. In the **New Bean Service** wizard, select the **Interface** link. This will open the **New Java Interface** wizard.

d. In the **New Java Interface** wizard, set the interface name to **GreetingService** and select **Finish**. This will take you back to the previous wizard.

e. Select **Finish** and save your changes.

Your project now has a *component* and a *component service*, represented by the blue rectangle and small green chevron, respectively. Given that this is a Bean implementation (indicated by the coffee bean icon), two new files have been added to your project. The **GreetingServiceBean** class corresponds to the component and the **GreetingService** interface corresponds to the component service.



This design is captured by the following underlying XML:

```
<sca:component name="GreetingServiceBean">
  <bean:implementation.bean
class="com.example.switchyard.switchyard_example.GreetingServiceBean
"/>
    <sca:service name="GreetingService">
      <sca:interface.java
interface="com.example.switchyard.switchyard_example.GreetingService
"/>
    </sca:service>
</sca:component>
```

3. **Customize the Bean.**

   a. Double-click on the component service (the small green chevron) and add the following **`greet()`** method.

```
package com.example.switchyard.switchyard_example;

public interface GreetingService {

 void greet();
}
```

   b. Save your changes to the **`GreetingService`** file.

   c. Now double-click on the **`GreetingServiceBean`** component and update the code as

follows.

```
package com.example.switchyard.switchyard_example;

import org.switchyard.component.bean.Service;

@Service(GreetingService.class)
public class GreetingServiceBean implements GreetingService {

    private int times;

    @Override
    public final void greet() {
        System.out.println((times++) + ". Service executed.");
    }

}
```

d. Save your changes to the **GreetingServiceBean** file.

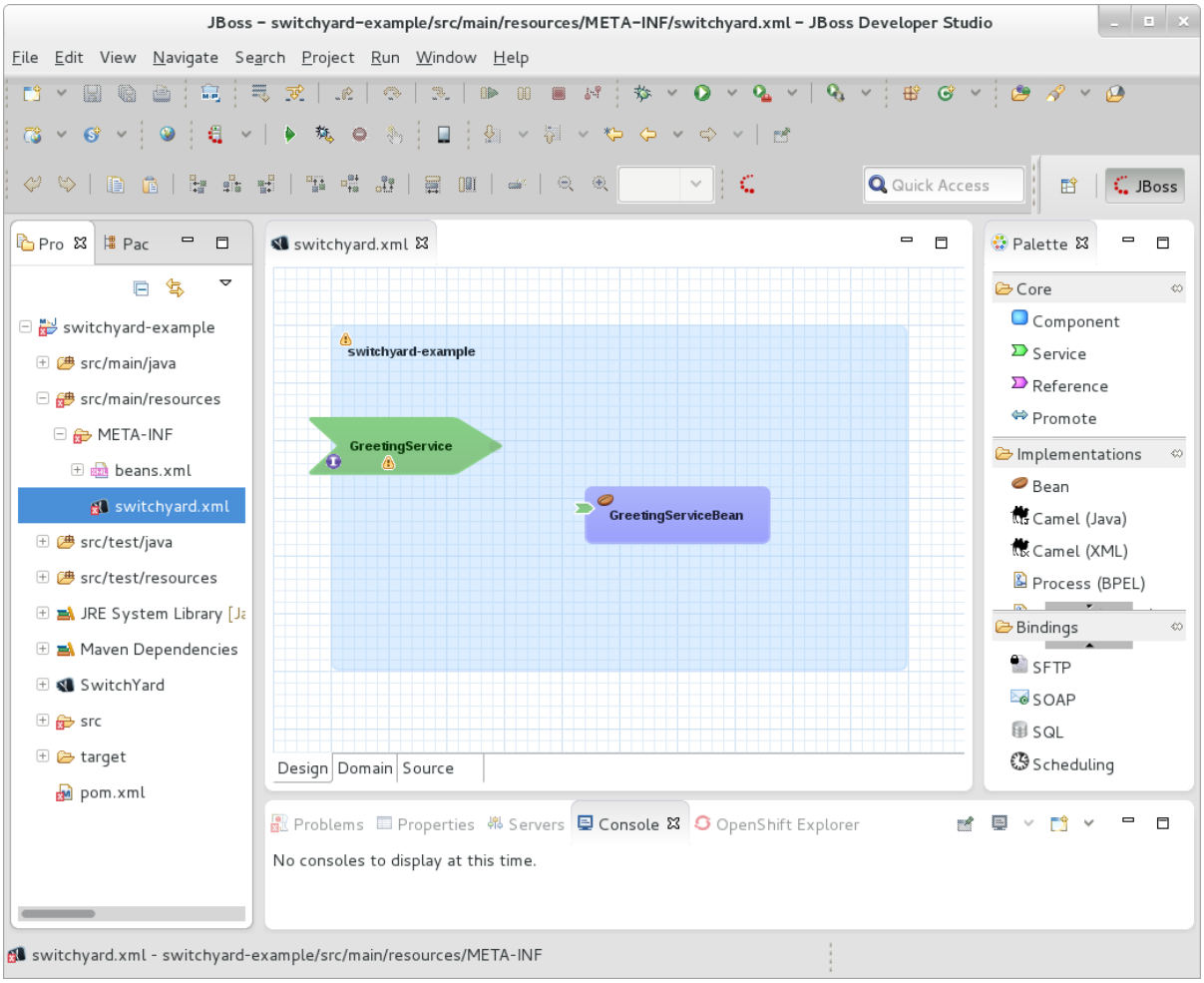
4. **Add a Service.**

a. Click and drag a **Service** from the SwitchYard **Palette** onto your composite. This will automatically invoke the **New Service** wizard.

b. In the **New Service** wizard, select **Browse**. Then choose the **GreetingService** interface which was created in the previous step, and select **OK**.

c. Select **Finish** and save your changes to the **switchyard.xml** file.

You have now created a *composite service*, represented by the large, green chevron.



The following entry has been added to the underlying XML:

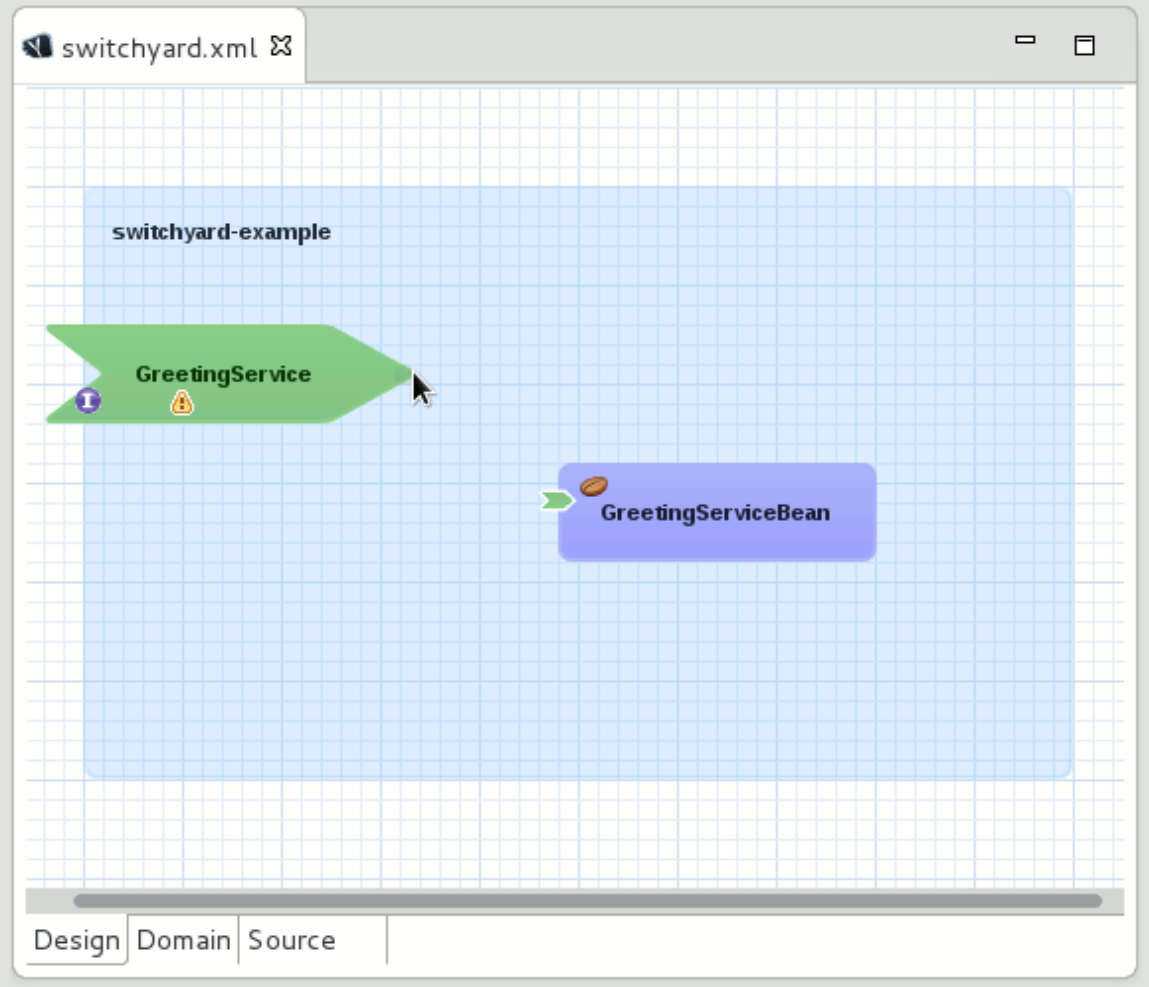
```
<sca:service name="GreetingService">
  <sca:interface.java
interface="com.example.switchyard.switchyard_example.GreetingService
"/>
</sca:service>
```

NOTE

At this point your project will report an error: Attribute 'promote' must appear on element 'sca:service'. This is because a composite service can only be realized by promoting a component service within the application. This is resolved in the next step.

5. **Promote the component service.**

   a. Click on the tip of the large green chevron (representing the composite service), and drag your cursor to connect it to the tail of the smaller green chevron (representing the component service).



b. Save your changes.

The component service is now promoted, with the following update to the XML:

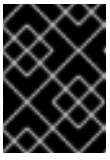
```
<sca:service name="GreetingService"
promote="GreetingServiceBean/GreetingService">
  <sca:interface.java
interface="com.example.switchyard.switchyard_example.GreetingService
"/>
</sca:service>
```

6. **Add a Scheduling binding.**

a. Click and drag a **Scheduling** binding from the SwitchYard **Palette** onto your composite service (the large green chevron). You will be prompted to provide **Scheduling Binding Details**.

b. Enter a name for your binding (we are using **EverySecondJob**) and enter the following value for **Cron**.

```
* * * * * ?
```

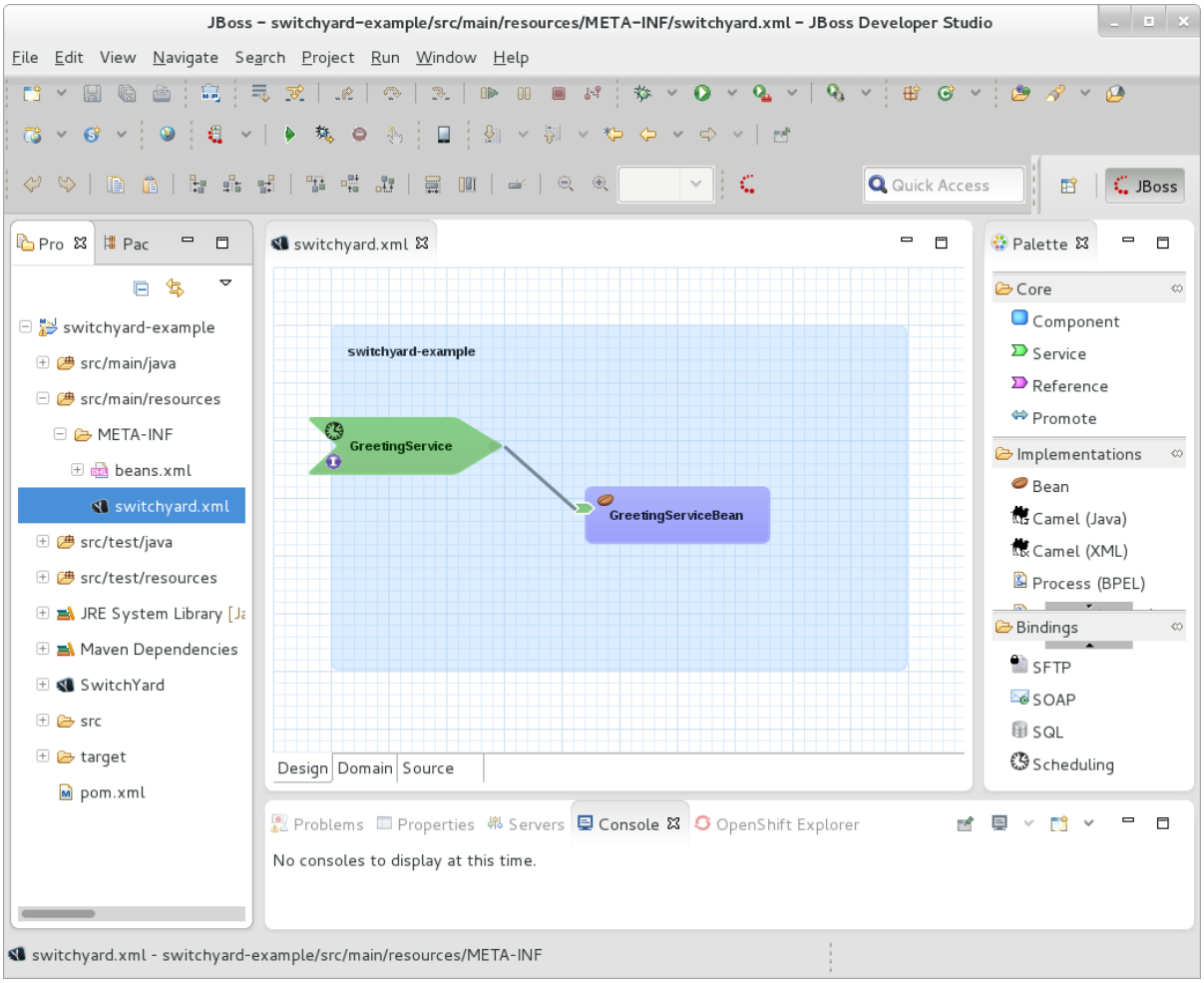


IMPORTANT

Ensure there is a single space between each of the characters above. Otherwise, the binding will not be created successfully.

c. Select **Finish** and save your changes.

The scheduling icon now appears on the composite service:



The following XML is added within the **sca:service** element accordingly:

```
<quartz:binding.quartz name="EverySecondJob">
  <quartz:name>EverySecondJob</quartz:name>
  <quartz:cron>* * * * * ?</quartz:cron>
</quartz:binding.quartz>
```

7. **Build and deploy the application.**
You are now ready to build and deploy your application.

a. From within the application's top directory (containing the application's **pom.xml** file), build the application with the following command:

```
mvn clean install
```

b. Start the application server.

```
[me@localhost jboss-eap-6.1]$ ./bin/standalone.sh
```

c. Open a browser and navigate to http://localhost:9990/console/index.html.

d. Select **Server** → **Manage Deployments** from the **Runtime** view.

e. Select **Add** to open the **Create Deployment** wizard. Select **Browse** and then select the file to deploy from the project's **target** folder; for example, **~/workspace/switchyard-example/target/switchyard-example-0.0.1-SNAPSHOT.jar**. Select **Open**.

f. Select **Next** to proceed to Step 2, then select **Save**.

g. Your deployment has been added to the list of **Available Deployments** but now you need to enable it. Select the deployment from the list, select the **En/Disable** button and then **Confirm**.

Now that the application is deployed and running, the following output appears (and continues to do so every second) in the server terminal window:

```
16:06:43,020 INFO  [stdout] (DefaultQuartzScheduler-camel-1_Worker-
2) 1. Service executed.
16:06:44,019 INFO  [stdout] (DefaultQuartzScheduler-camel-1_Worker-
3) 2. Service executed.
16:06:45,009 INFO  [stdout] (DefaultQuartzScheduler-camel-1_Worker-
4) 3. Service executed.
16:06:46,009 INFO  [stdout] (DefaultQuartzScheduler-camel-1_Worker-
5) 4. Service executed.
16:06:47,010 INFO  [stdout] (DefaultQuartzScheduler-camel-1_Worker-
6) 5. Service executed.
16:06:48,011 INFO  [stdout] (DefaultQuartzScheduler-camel-1_Worker-
7) 6. Service executed.
...
```

Report a bug

8.3. CREATE A WEB SERVICE PROXY

Sometimes you want to inject some logic between a client and a web service, without changing the service itself. For example:

- If you are using multiple versions of a service, you need intelligent routing to figure out which version of the service a client is invoking.
- If the web service does not provide security on its own, you might need to add a layer of security.
- You might want additional auditing/logging.
- You might have to transform/manipulate messages to make them compatible with the service.
- You might want to change the backend service later, without forcing a change on the client.

You can do this with a web service proxy. A web service proxy sits between the client and the web service provider:

```
[client]  <--->  [web service proxy]  <--->  [web service]
```

The **camel-soap-proxy** quickstart application (in the ***EAP_HOME*/quickstarts/switchyard** directory) is provided as an example. You can follow the instructions in the provided **Readme.md** file to build and deploy this application.

The remainder of this section describes how to develop your own web service proxy.

Tutorial Overview

This tutorial consists of three parts:

1. Deploy and test a web service.
2. Develop a web service proxy that acts as a proxy to the web service.
3. Deploy and test the web service proxy.

By the end of this tutorial you will have the knowledge and skills necessary to develop, deploy and test a web service proxy for your own web service.

Prerequisites

- The JBoss Integration and SOA Development tools are installed for JBoss Developer Studio.
- Maven is installed and configured.
- Your application server is running (it is assumed, for this tutorial, on default port http://localhost:8080).

Deploy and Test a Web Service

The **`camel-soap-proxy`** quickstart application provides a suitable web service, **`ReverseService.war`**, for the purpose of this exercise. Given a SOAP message, this web service responds with the message text reversed. In this section you will build, deploy and test this web service.

1. From your ***`EAP_HOME`*`/quickstarts/switchyard/camel-soap-proxy`** directory, run the following command:

   ```
   $ mvn clean install
   ```

   This builds the quickstart application, including the web service required for this exercise: ***`EAP_HOME`*`/quickstarts/switchyard/camel-soap-proxy/target/ReverseService.war`**.

2. Deploy the service to your running application server by copying the **`ReverseService.war`** file to the server's **`deployments`** folder, ***`EAP_HOME`*`/standalone/deployments/`**.

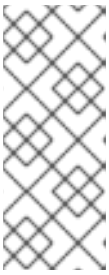
   The service is now ready to accept requests on http://localhost:8080/ReverseService.

3. Use your favorite SOAP client to test the service. If you send the following request:

   ```
   <SOAP-ENV:Envelope xmlns:SOAP-
   ENV="http://schemas.xmlsoap.org/soap/envelope/">
       <SOAP-ENV:Header/>
       <SOAP-ENV:Body>
           <ns2:reverse xmlns:ns2="urn:switchyard-quickstart:camel-
   soap-proxy:1.0">
               <text>Hello world!</text>
           </ns2:reverse>
       </SOAP-ENV:Body>
   </SOAP-ENV:Envelope>
   ```

   You receive the following response, with the message text reversed:

```
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <ns2:reverseResponse xmlns:ns2="urn:switchyard-
quickstart:camel-soap-proxy:1.0">
         <text>!dlrow olleH</text>
      </ns2:reverseResponse>
   </soap:Body>
</soap:Envelope>
```



NOTE

You can confirm the web service address (and retrieve the associated WSDL) using JBoss Management Console. Login to the console (default address is http://localhost:9990/console/). From the **Runtime** view, navigate to **Manage Deployments**. Select **ReverseService.war** → **webservices** → **ReverseService**. Note the URL provided for **WSDL**.

Develop the Web Service Proxy

Now that the web service has been deployed on the server, develop the proxy to interface to it.

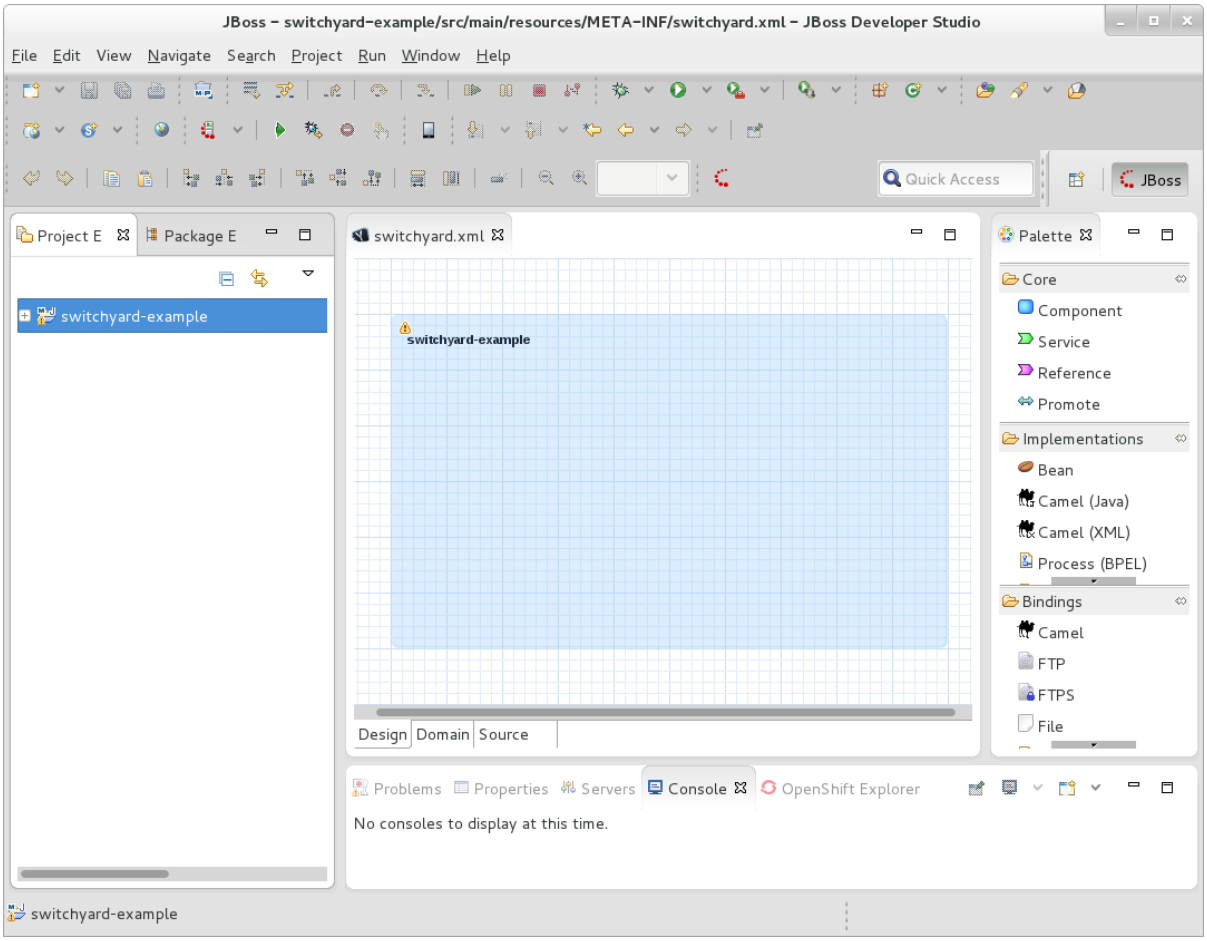
1. **Create a new SwitchYard project in JBoss Developer Studio.**

   a. Select **File** → **New** → **Project**. Then select the appropriate wizard: **SwitchYard** → **SwitchYard Project** and select **Next**.

   b. Choose a project name (we are using **switchyard-example**) and select **Next**.

   c. Select **Finish** to proceed.

   At this point, the new project is generated and appears in the **Project Explorer** view.



2. **Add the WSDL file for the web service as a resource.**

   a. Copy the WSDL file for the deployed service to **`.../switchyard-example/src/main/resources/META-INF`**. You can do this using **`wget`**. For example, on the command line, from within the application's **`META-INF`** directory, run:

   ```
   wget http://localhost:8080/ReverseService?wsdl -O
   ReverseService.wsdl
   ```

   b. In JBoss Developer Studio, right-click on the project in the **`Project Explorer`** view, and select **`Refresh`**. Navigate the project files to confirm the file has been added as a resource.

3. **Add a Camel (XML) implementation.**

   a. Open the **`Design`** tab for the **`switchyard.xml`** file.

   b. Click and drag a **`Camel (XML)`** implementation from the SwitchYard **`Palette`** onto the composite (the blank canvas). This invokes a wizard to help you create a new Camel route file resource. If the **`Palette`** is not yet visible, select **Window** → **Show View** → **Other**, then expand **`General`** and select **`Palette`**.

   c. In the wizard, select the **`WSDL`** option, and browse to select the **`ReverseService.wsdl`** file for the interface.

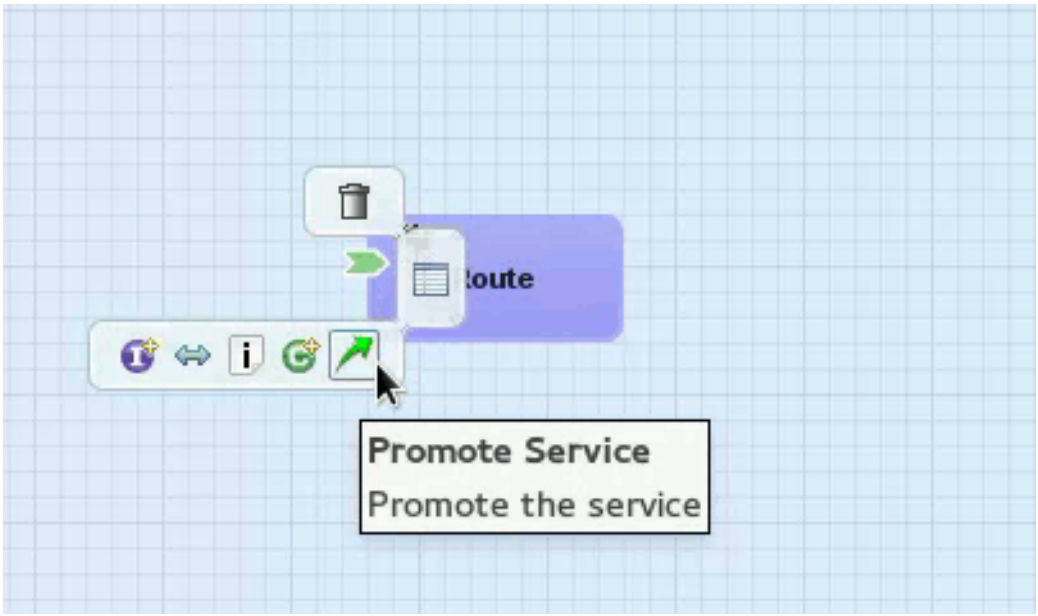
   d. Change the **`Service Name`** to **`ProxyService`**.

   e. Select **`Finish`** and save the project.

Your project now has a *component* and a *component service*, represented by the blue rectangle and small green chevron, respectively. A **`route.xml`** file has been added to your project, corresponding to the component. The **`ReverseService.wsdl`** corresponds to the component service. If you double-click on either of these graphical entities, the corresponding file opens for viewing.

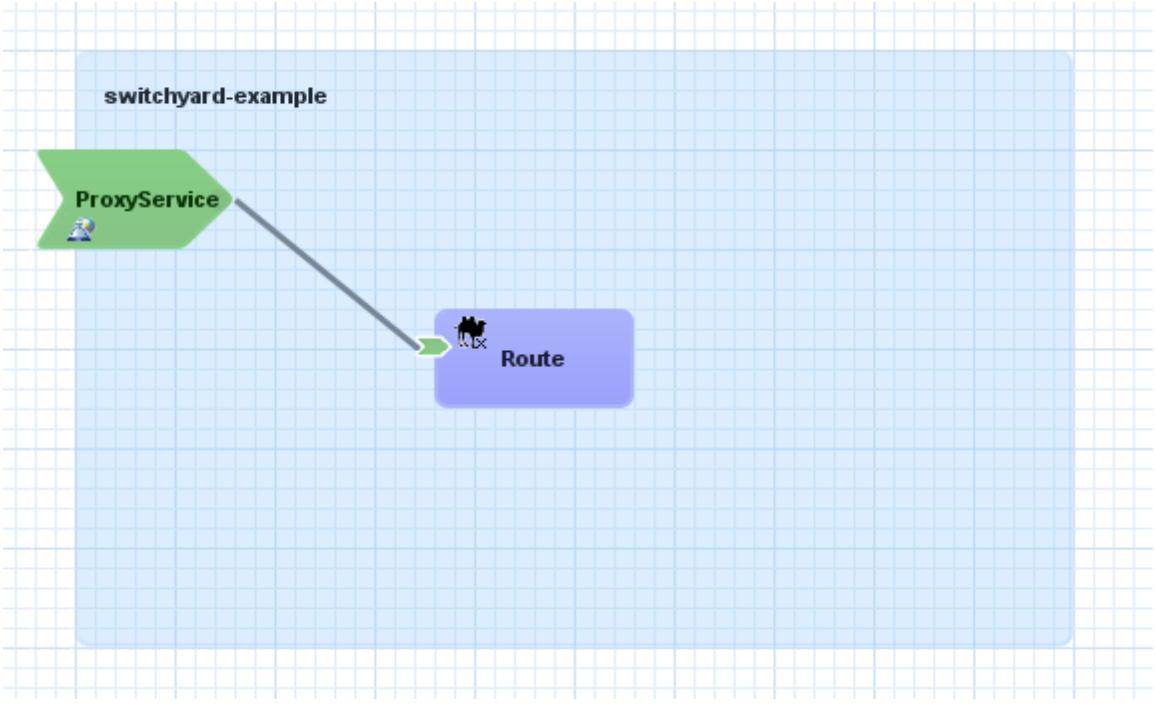
4. **Promote the component service.**

   a. Move your cursor over the component service and select **`Promote Service`**.



   b. You are prompted to provide details for the promotion. Leave the default values. Select **`Finish`**, and save the project.

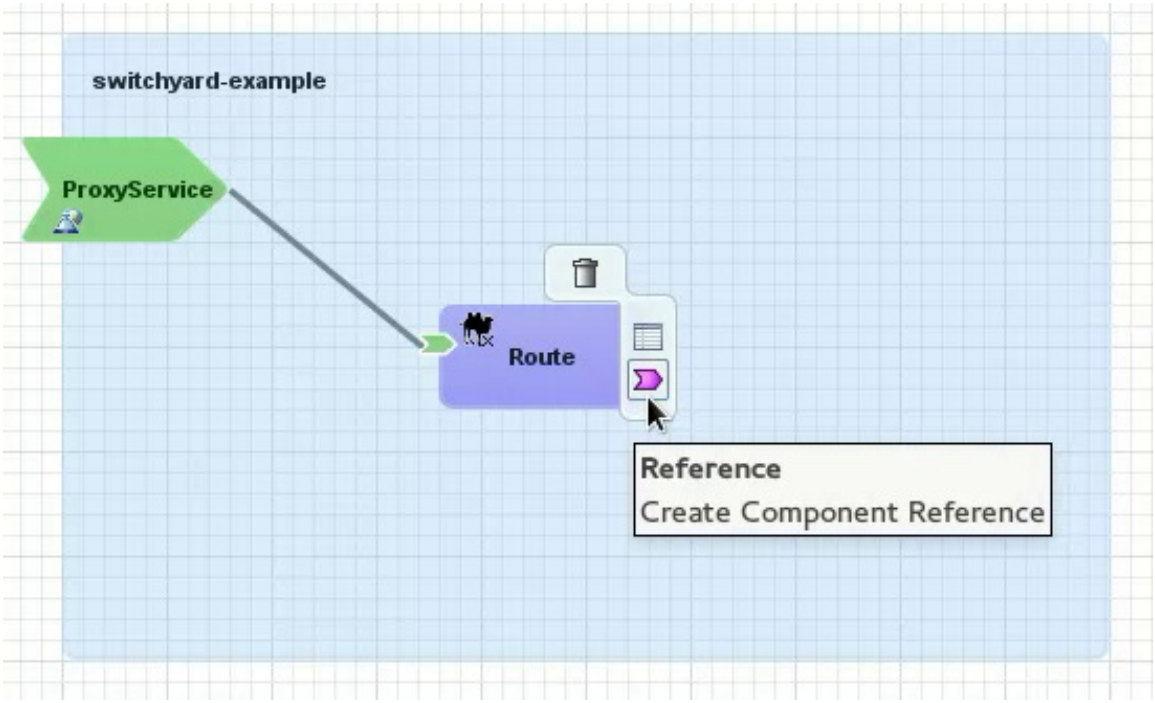
   Your component service has now been promoted to a corresponding *composite service* (represented by the large green chevron):



5. **Create a component reference.**

   a. Move your cursor over the component and select **`Reference`**

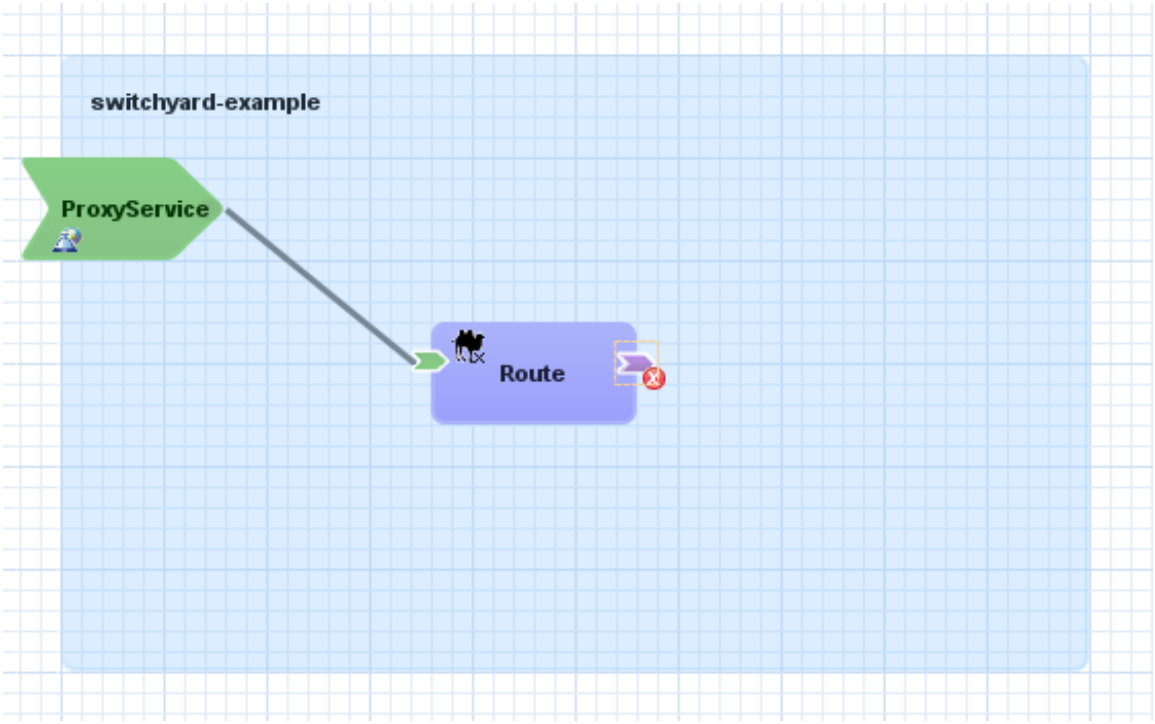
a. Move your cursor over the component and select **Reference**.



b. You are prompted to provide details for the new reference. Select the **WSDL** option, and browse to select the **ReverseService.wsdl** file for the interface. Leave the default **Service Name** as **ReverseService**.

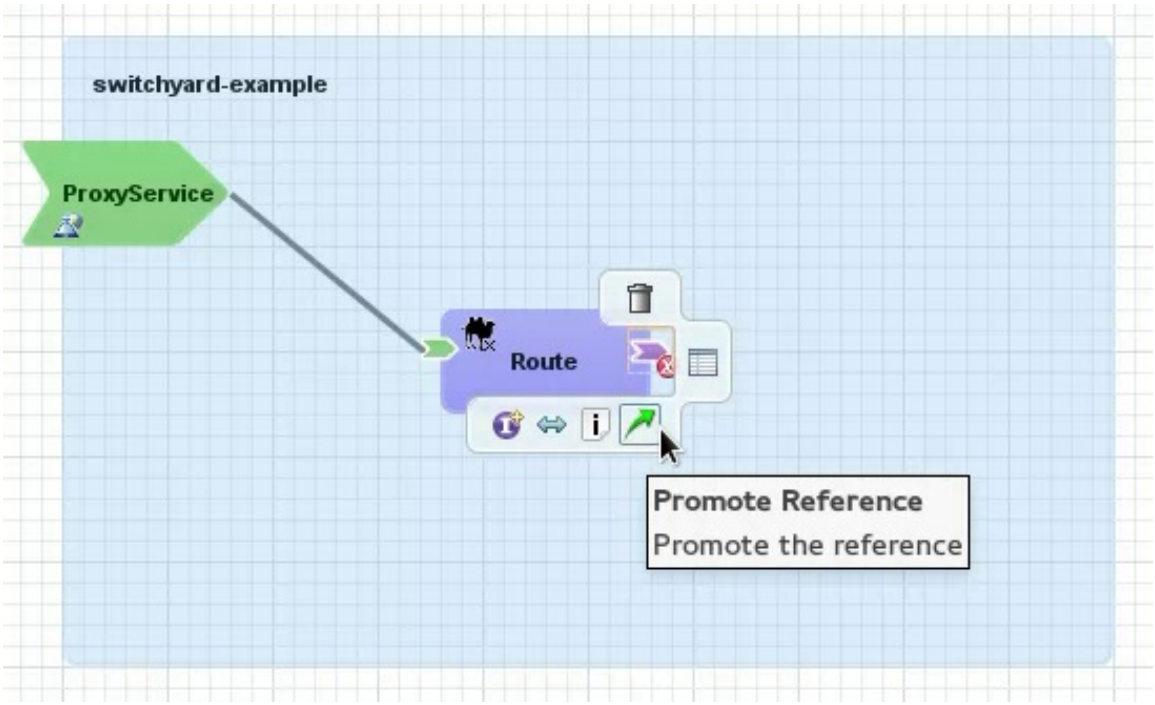
c. Select **Finish** and save the project.

Your component now has a *component reference* (small purple chevron). An error appears until it is promoted in the next step.



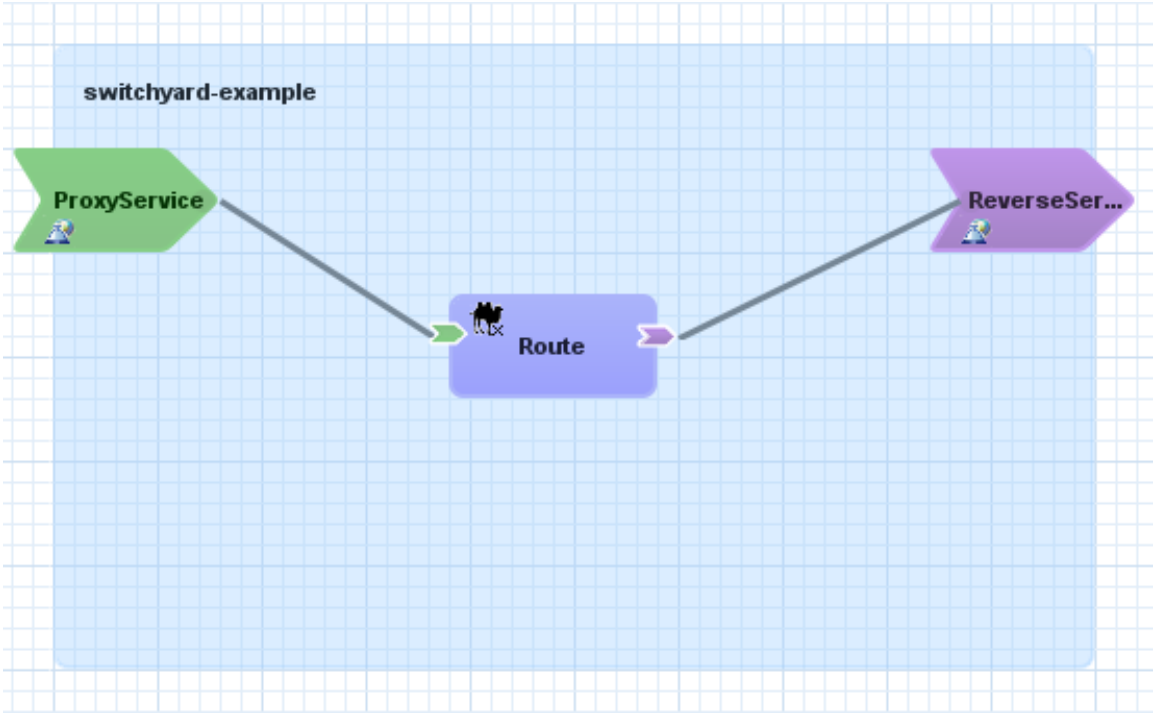
6. **Promote the component reference.**

a. Move your cursor over the component reference and select **Promote Reference**.



b. You are prompted to provide details for the promotion. Leave the default values. Select **`Finish`**, and save the project.

   Your component reference has now been promoted to a corresponding *composite reference* (represented by the large purple chevron):



7. **Create a route from the proxy endpoint to the web service endpoint.**

   a. Open the **`route.xml`** file. You can open it from the **`Project Explorer`** or by double-clicking on the Camel **`Route`** component. By default it opens in the **`Design`** view. Select the **`Source`** view to expose the Spring XML.

   b. From looking at the XML, you can see that, currently, any traffic received from the proxy service will be logged.

To forward this traffic on to the web service, add the web service as an endpoint:

```
<?xml version="1.0" encoding="ASCII"?>
<routes xmlns="http://camel.apache.org/schema/spring">
  <route>
    <from uri="switchyard://ProxyService"/>
    <log message="ProxyService - message received: ${body}"/>
    <to uri="switchyard://ReverseService"/>
  </route>
</routes>
```

c. Save your changes.

8. **Add a SOAP binding to the composite service.**

   a. Drag a SOAP binding from the SwitchYard **`Palette`** onto the composite service (**`ProxyService`**).

   b. You are prompted for SOAP binding details. Set the following values, leaving all other default values:

Property	Value
Name	**`soap-proxy`**
Context path	**`proxy`**
Server Port	**`:${webPort}`**

   c. Select **`Finish`** and save your changes.

9. **Add a SOAP binding to the composite reference.**

   a. Drag a SOAP binding from the SwitchYard **`Palette`** onto the composite reference (**`ReverseService`**).

   b. You are prompted for SOAP binding details. Set the following values, leaving all other default values:

Property	Value
Name	**`soap-service`**
Endpoint Address	**`http://localhost:${webPort}/ReverseService`**

   c. Select **`Finish`** and save your changes.

10. **Define the value of `webPort`.**

    a. From the **`Domain`** view of the **`switchyard.xml`** file, add a domain property with the following values:

following values:

Name	Value
`webPort`	`${org.switchyard.component.http.standalone.port:8080}`

b. Save your changes.

Your web service proxy application is now ready to build and deploy.

Deploy and Test the Web Service Proxy

1. **Build and deploy the web service proxy.**

   a. From within the application's top directory (containing the application's `pom.xml` file), build the application with the following command:

   ```
   mvn clean install
   ```

   b. Open a browser and navigate to http://localhost:9990/console/index.html.

   c. Select **Server** → **Manage Deployments** from the `Runtime` view.

   d. Select `Add` to open the `Create Deployment` wizard. Select `Browse` and then select the file to deploy from the project's `target` folder; for example, `~/workspace/switchyard-example/target/switchyard-example-0.0.1-SNAPSHOT.jar`. Select `Open`.

   e. Select `Next` to proceed to Step 2, then select `Save`.

   f. Your deployment has been added to the list of `Available Deployments` but now you need to enable it. Select the deployment from the list, select the `En/Disable` button and then `Confirm`.

2. **Test your application.**
   Your web service proxy is now ready to accept requests on http://localhost:8080/proxy/ReverseService.

   a. Use your favorite SOAP client to send a message to test the proxy service in the same way as you did when you tested the web service, only use the proxy address instead. After sending a message, you receive a response with the text reversed, the same as when it was sent directly to the web service. However, when sending to the proxy, you can see that the message is also logged on the server as prescribed by the `route.xml` in the proxy service; for example:

   ```
   11:50:29,100 INFO  [route2] (http-/127.0.0.1:8080-14)
   ProxyService - message received:
    <ns2:reverse xmlns:ns2="urn:switchyard-quickstart:camel-soap-
   proxy:1.0">
     <text>Hello world!</text>
    </ns2:reverse>
   ```

CHAPTER 9. GETTING STARTED WITH CAMEL ON JBOSS EAP

9.1. CAMEL CXF EXAMPLE

This tutorial takes a standard Camel CXF example and shows you how to deploy it into a Web server, by packaging the application as a WAR. In this example, the Web service is implemented by binding the service to a Camel route using the Camel CXF component.

Prerequisites

Maven 3.0.x must be installed and configured.

1. Open a terminal and navigate to the ***EAP_HOME*/quickstarts/camel/camel-example-cxf-tomcat** directory.

2. Run the following command to build and package the example:

   ```
   mvn clean install
   ```

   If this command executes successfully, the application will be packaged in ***EAP_HOME*/quickstarts/camel/camel-example-cxf-tomcat/target/camel-example-cxf-tomcat.war**.

3. If your application server is not running. Start it now. You can check if your server is running by opening a browser and navigating to http://localhost:8080. If you can login successfully to the management console, the server is running.

4. Deploy the **camel-example-cxf-tomcat** example to the running server by copying the **camel-example-cxf-tomcat.war** file to the server's deployment directory, ***EAP_HOME*/MODE/deployments**.

5. You can use a Web browser to query the WSDL contract from the newly deployed Web service. Navigate to the following URL:

   http://localhost:8080/camel-example-cxf-tomcat/webservices/incident?wsdl

6. Run the test client against the deployed Web service. From the quickstart directory, enter the following command:

   ```
   mvn exec:java
   ```

   If the client runs successfully, you should see some output like the following in your command window:

   ```
      ...
   [INFO] --- exec-maven-plugin:1.1.1:java (default-cli) @ camel-
   example-cxf-tomcat ---
   2013-07-24 13:59:16,829 [teClient.main()] INFO
   ReflectionServiceFactoryBean
   - Creating Service
   {http://incident.cxf.example.camel.apache.org/}IncidentService
   from class org.apache.camel.example.cxf.incident.IncidentService
   OK;123
   ```

```
IN PROGRESS
[INFO] ------------------------------------------------------------
--------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------
--------------
[INFO] Total time: 7.445s
[INFO] Finished at: Wed Jul 24 13:59:17 CEST 2013
[INFO] Final Memory: 10M/81M
[INFO] ------------------------------------------------------------
--------------
```

7. If you have an interface such as soapUI installed, you can test the service using requests such as the following sample SOAP requests:

```
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
        <ns1:reportIncident
xmlns:ns1="http://incident.cxf.example.camel.apache.org/">
            <arg0>
                <details>blah blah</details>
                <email>davsclaus@apache.org</email>
                <familyName>Smith</familyName>
                <givenName>Bob</givenName>
                <incidentDate>2011-11-25</incidentDate>
                <incidentId>123</incidentId>
                <phone>123-456-7890</phone>
                <summary>blah blah summary</summary>
            </arg0>
        </ns1:reportIncident>
    </soap:Body>
</soap:Envelope>
```

```
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
        <ns1:statusIncident
xmlns:ns1="http://incident.cxf.example.camel.apache.org/">
            <arg0>
                <incidentId>456</incidentId>
            </arg0>
        </ns1:statusIncident>
    </soap:Body>
</soap:Envelope>
```

NOTE

Detailed instructions to run each quickstart are documented in the **`README.txt`** file located in that quickstart's directory.

Report a bug

9.2. CAMEL SERVLET EXAMPLE

This tutorial takes a standard Apache Camel example and shows you how to deploy it into a Web server, by packaging the application as a WAR.

Prerequisites

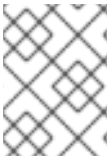
Maven 3.0.x must be installed and configured.

1. Open a terminal and navigate to the ***EAP_HOME*/quickstarts/camel/camel-example-servlet-tomcat** directory.
2. Run the following command to build and package the example:

   ```
   mvn clean install
   ```

   If this command executes successfully, the application will be packaged in ***EAP_HOME*/quickstarts/camel/camel-example-servlet-tomcat/target/camel-example-servlet-tomcat-*VERSION*.war**.
3. If your application server is not running. Start it now. You can check if your server is running by opening a browser and navigating to http://localhost:8080. If you can login successfully to the management console, the server is running.
4. Deploy the **camel-example-servlet-tomcat** example to the running server by copying the **camel-example-servlet-tomcat-*VERSION*.war** file to the server's deployment directory, ***EAP_HOME*/MODE/deployments**.
5. Navigate to the following URL in your browser (assuming *VERSION* is **2.10.0.redhat-60024**):

   http://localhost:8080/camel-example-servlet-tomcat-2.10.0.redhat-60024/
6. You can test the servlet by following the link in the text: "To get started click this link."



NOTE

Detailed instructions to run each quickstart are documented in the **README.txt** file located in that quickstart's directory.

Report a bug

APPENDIX A. QUICKSTART APPLICATIONS

A.1. ABOUT QUICKSTART APPLICATIONS

Each quickstart application demonstrates a specific piece of JBoss Fuse Service Works functionality to help you build services for your solution. All of the quickstart applications are included in the ***EAP_HOME*/quickstarts** directory of your installation.

NOTE

Instructions to run each quickstart application are documented in a README file located in the application directory.

Report a bug

A.2. SAMPLE SWITCHYARD QUICKSTARTS

The following list is a sample of the SwitchYard quickstarts available within the **EAP_HOME/quickstarts/switchyard** directory:

remote-invoker

The RemoteInvoker serves as a remote invocation client for SwitchYard services. It allows non-SwitchYard applications to invoke any service in SwitchYard which uses a <binding.sca> binding. It is also used by the internal clustering channel to facilitate intra-cluster communication between instances.

bean-service

Implements a service using a CDI bean and exposes it through a SOAP gateway.

bpm-service

Uses BPMN 2 to provide and utilize SwitchYard services.

camel-service

This is a basic routing example that uses XML and Java DSL Camel routes.

camel-jms-binding

Demonstrates how a Camel component can be used as a SwitchYard gateway.

demos/orders

Launches a web application which invokes a SwitchYard service from a JSF.

NOTE

You can find additional quickstarts by going to the **quickstarts** directory.

Report a bug

A.3. USING SWITCHYARD QUICKSTARTS

Prerequisites

- Maven 3.0.x must be installed.

NOTE

Run the following command to verify that Maven is installed successfully on your machine: **`mvn --version`**

Procedure A.1. Task

1. Open a terminal.
2. Navigate to a quickstart directory under ***`EAP_HOME`*`/quickstarts/switchyard`** where the project **`pom.xml`** file for the quickstart application is located. For example:

```
EAP_HOME/quickstarts/switchyard/remote-invoker
```

3. Run the following command:

```
mvn clean install
```

Result

The quickstart compiles and is ready for use.

Report a bug

A.4. RUNNING QUICKSTARTS FROM JBOSS DEVELOPER STUDIO

Overview

This topic demonstrates how to import a quickstart application to JBoss Developer Studio and then deploy it to a running application server.

Prerequisites

The JBoss Integration and SOA Development tools must be installed from the JBoss Developer Studio Integration Stack.

Procedure A.2. Task

1. Open JBoss Developer Studio.
2. Click **File** → **Import** → **Maven** → **Existing Maven Projects**.
3. Select **`Browse`** and navigate to the quickstart directory, for example, ***`EAP_HOME`*`/quickstarts/switchyard/bean-service`** and then select **`OK`**.

   The import tool will scan the directory to locate the associated **`pom.xml`** file.
4. Select **`Finish`**.

5. The quickstart is listed in the Project Explorer tab. You can expand the project to explore its contents.

6. In the Project Explorer tab, right-click on the project's name and click **Run as** → **Run on server** → **EAP**.

Result

The quickstart application is deployed to the server and will be enabled by default.

Report a bug

APPENDIX B. PREREQUISITE SOFTWARE

B.1. INSTALL OPEN JDK ON RED HAT LINUX

This topic covers the steps to install Open JDK on RedHat Linux.

1. Subscribe to the Base Channel Obtain the OpenJDK from the RHN base channel. (Your installation of Red Hat Enterprise Linux is subscribed to this channel by default.)
2. Install the Package. Use the *yum* utility to install OpenJDK: **`yum install java-1.7.0-openjdk-devel`**
3. Verify that Open JDK is now your system default. You can ensure the correct JDK is set as the system default by following the steps below.
4. a. As a root user, run the alternatives command for java: **`/usr/sbin/alternatives --config java`**

   b. Select **`/usr/lib/jvm /jre-1.7.0-openjdk/bin/java`** .

   c. Apply the same for javac: **`/usr/sbin/alternatives --config javac`**

   d. Select **`/usr/lib/jvm /java-1.7.0-openjdk/bin/java.`**

Result

Open JDK is installed successfully on your machine.

Report a bug

B.2. INSTALL MAVEN

Prerequisites

The following software must be installed:

- An archiving tool for extracting the contents of compressed files.
- Open JDK.

Procedure B.1. Install Maven

1. **Download Maven.**

   a. Enter http://maven.apache.org/download.cgi in the address bar of a browser.

   b. Download **`apache-maven-3.0.5`** ZIP file and save it to your hard drive.

2. **Install and configure Maven.**

   - **On Red Hat Enterprise Linux**

     a. Extract the ZIP archive to the directory where you wish to install Maven.

     b. Open a terminal.

c. Add the M2_HOME environment variable by entering the following command:

```
export M2_HOME=/usr/local/apache-maven/apache-maven-3.0.x
```

d. Add the M2 environment variable by entering the following command:

```
export M2=$M2_HOME/bin
```

e. Add the M2 environment variable to your path by entering the following command:

```
export PATH=$M2:$PATH
```

f. Make sure that JAVA_HOME is set to the location of your JDK. For example:

```
export JAVA_HOME=/usr/lib/jvm/java-1.7.0-openjdk.x86_64
```

g. Make sure that $JAVA_HOME/bin is in your PATH environment variable.

h. Run the following command to verify that Maven is installed successfully on your machine:

```
mvn --version
```

- **On Microsoft Windows**

a. Extract the ZIP archive to the directory where you wish to install Maven. The subdirectory **`apache-maven-3.0.x`** is created from the archive.

b. Press **`Start+Pause|Break`**. The **`System Properties`** dialog box is displayed.

c. Click the **`Advanced`** tab and click **`Environment Variables`**.

d. Under System Variables, select **`Path`**.

e. Click **`Edit`** and add the two Maven paths using a semicolon to separate each entry.

- Add the M2_HOME variable and set the path to **`C:\Program Files\Apache Software Foundation\apache-maven-3.0.X`**.
- Add the M2 variable and set the value to %M2_HOME%\bin.

f. Update or create the Path environment variable:

- Add the %M2% variable to allow Maven to be executed from the command line.
- Add the variable %JAVA_HOME%\bin to set the path to the correct Java installation.

g. Click **`OK`** to close all the dialog boxes including the **`System Properties`** dialog box.

h. Open Windows command prompt and run the following command to verify that Maven is installed successfully on your machine:

```
mvn --version
```

Result

Maven is successfully installed and configured on your machine.

Report a bug

APPENDIX C. REVISION HISTORY

Revision 6.0.0-70 **Wednesday July 22 2015** **B Long**
Added note about how enabling the Java Security Manager may reduce server performance..

Revision 6.0.0-69 **Friday June 5 2015** **B Long**
Added note about setting PATH and other variables.

Revision 6.0.0-68 **Tuesday April 14 2015** **Petr Penicka**
Improved the Creating the Camel Quartz Binding Quickstart Application chapter.

Revision 6.0.0-67 **Thursday March 19 2015** **B Long**
Adding getting started tutorial on how to create a web service proxy (FUSEDOC-318).

Revision 6.0.0-66 **Monday February 23 2015** **B Long**
Version number updated from 6 to 6.0 for consistency.

Revision 6.0.0-65 **Wednesday December 17 2014** **B Long**
Adding animated GIFs to Camel Quartz Binding quickstart topic as per Bug 1170393.

Revision 6.0.0-64 **Wednesday December 10 2014** **Anshu Mahajan**